Final Technical Report 2092

APPLICATION OF EXPERT SYSTEMS IN PROJECT MANAGEMENT DECISION AIDING

NAS5-30040

August 1987

SCIENTIFIC AND TECHNICAL REPORT

Submitted to:

National Aeronautics and Space Administration
Eugene Grunby
Contracting Officer's Technical Representative
Goddard Space Flight Center
Greenbelt, MD

Prepared by:

Regina Harris
Steven Shaffer
James Stokes
David Goldstein

®

SUMMARY

The purpose of this research was to assess the feasibility of developing an expert systems-based project management decision aid to enhance the performance of NASA project managers. The research effort included extensive literature reviews in the areas of project management, project management decision aiding, expert systems technology, and human-computer interface engineering. Literature reviews were augmented by focused interviews with NASA managers. Time estimation for project scheduling was identified as the target activity for decision augmentation, and a design was developed for an Integrated NASA System for Intelligent Time Estimation (INSITE). The proposed INSITE design was judged feasible with a low level of risk. A partial proof-of-concept experiment was performed and was successful.

Specific conclusions drawn from the research and analyses include: time estimation is critical to NASA planning/scheduling needs; existing methods of project time estimation rely on the identification of analogous projects which have already been completed; the identification of analogies is usually based on the experience and recall of a single person, severely limiting the number and accuracy of those project histories which can be brought to bear on the problem; expertise in estimation by analogy can be captured using existing technologies such as pattern matching, statistical inference, and rule-based analysis; given an effective human-computer interface, an automated estimation-by-analogy system will enhance manager performance by providing a framework for systematic time estimation and by allowing managers to transcend the limitations of their own experience.

The INSITE concept is potentially applicable in any management sphere, commercial or government, where time estimation is required for project scheduling. As project scheduling is a nearly universal management activity, the range of possibilities is considerable. The INSITE concept also holds potential for enhancing other management tasks, especially in areas such as cost estimation, where estimation-by-analogy is already a proven method.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

1.0 INTRODUCTION

1.1 PHASE I OBJECTIVES

This report was produced as part of a Phase I research effort awarded by the National Aeronautics and Space Administration (NASA) under the Small Business Innovation Research (SBIR) Program, contract NAS5-30040, monitored by the Goddard Space Flight Center.

Development of complex systems, such as the Space Shuttle or Space Station, requires managing and performing many activities simultaneously (e.g., design, construction, and deployment of the various subsystems). Inadequate management of these activities can result in critical and costly delays. The primary objective of the Phase I research was to investigate the feasibility of developing a project management aid based on expert systems technology to assist NASA project managers in the successful management of complex programs.

The original Phase I proposal outlined four major tasks to be conducted:

1. Identification of current expert systems technology relevant to project management.

2. Identification of stand-alone NASA project management tasks which are well-defined but also complex or require a large measure of expertise.

3. Assessment of the feasibility of a project management expert systems tool, including development of a system architectural concept.

4. Documentation of the Phase I efforts as well as development of a work plan for Phase II.

All objectives for the Phase I research effort have been met. This report presents the documentation of these tasks.

1.2 RESULTS OF THE PHASE I RESEARCH EFFORT

In an environment of limited resources but with an almost boundless horizon of new ideas, goals, proposed projects, and experiments, it is necessary to utilize the available resources in a maximally efficient manner. This is especially true for an organization such as NASA, in which important proposed projects require substantial funding. A system to identify and store relevant information on previous and ongoing programs that will allow project managers to accurately estimate important program milestones such as time-to-complete is needed. Successful project managers have developed intuitive associations for interconnecting seemingly isolated facts and propositions about a project as well as methods for determining what are the important aspects and interrelationships between project variables. The goal of this research effort is to determine the feasibility of capturing and formalizing this knowledge into an expert system for project managers.

Based upon our interviews with NASA personnel, time estimation for project scheduling was identified as the focus for our continuing effort under the Phase I SBIR. It is an area of project management which is both critical to NASA's needs and amenable to the application of expert systems technology.

One of the main problems currently being faced at the Goddard Space Flight Center is that of scheduling the design and construction of the Space Station. The scheduling that is required for this effort is of the Gantt and CPM/PERT chart type. Basically, this is the process of decomposing a project into tasks and designing a structure which graphically represents the dependencies and interactions of those tasks. Discussions with NASA personnel revealed that project management software, delivered by Boeing under the Technical and Management Information System (TMIS) effort, is expected to provide the capability to generate project management charts (e.g., Gantt, CPM, and PERT), determine critical paths, and assist in monitoring project activities and propagating the effects of schedule changes. However, this is a customized, expensive, and sophisticated data base that will require extensive training and will only be available to a limited number of priority programs. Therefore, a need exists for a cost-effective, user-friendly system for project management that is available to a large user community and requires minimal training.

The Phase I effort concentrated on an investigation of an automated system to assist project managers in the estimation of the time requirements for tasks. The proposed system, the Integrated NASA System for Intelligent Time Estimation (INSITE), utilizes analogous cases and historical data to develop accurate time estimates. The INSITE inference mechanism will be based upon recent advances in artificial intelligence (i.e., reasoning by analogy, pattern matching, etc.). As with any expert system shell, INSITE will require some customization, based on historical project information, for each installation site. The more information available, the better the INSITE system will perform. An elicitation tool to help extract information from the human operator and to increase INSITE's knowledge base was also investigated.

The following steps served as the approach to determining the feasibility of INSITE:

1. Review relevant technology in reasoning by analogy, pattern matching, statistical inference, and knowledge elicitation.

2. Review project scheduling methodologies and cost estimation techniques (as a model for time estimation).

3. Specify system functionality for the time estimation aid, including what the system will do for the user, what the system will demand of the user, and how the system will look to the user.

4. Review relevant human factors engineering literature to determine relevant techniques for the user-computer interface.

5. Evaluate the feasibility of constructing a system as described, taking into account such factors as current technology, cost/benefit trade-offs, etc.

6. Develop a preliminary design for the Phase II prototype effort.

The development of the INSITE system will assist NASA program and project managers in planning and scheduling projects. Computerized time estimation systems, such as the proposed INSITE system, assist in formalizing the time estimation process since all project managers will have access to the entire body of knowledge and not just one isolated piece. The use of a system such as INSITE will result in improved and more consistent time estimates, particularly by novice project managers. INSITE will also increase the productivity of the project manager by automating the lengthy and tedious procedures currently used to determine time estimates. It will also enhance responsiveness to changing requirements by allowing the project manager to modify project features and quickly obtain new estimates.

1.3 REPORT OVERVIEW

This report is organized as follows:

- Section 2 presents an overview of the project management process.
- Section 3 presents the technical approach used to develop the INSITE system.
- Section 4 provides a review of the research conducted on technologies relevant for the INSITE estimator.
- Section 5 provides a review of the research conducted on technologies relevant for the INSITE knowledge base.
- Section 6 provides a review of the research conducted on technologies relevant for the INSITE user interface.

- Section 7 presents the preliminary design of the INSITE system.

- Section 8 presents an assessment of the proposed system's feasibility, including risk areas, and provides a description of the goals and objectives of a Phase II program to develop the INSITE system.

2.0 PROJECT MANAGEMENT

Project management (PMGT) is defined as: "the coordination of group activity wherein the manager plans, organizes, staffs, directs, and controls to achieve an objective with constraints on time, cost, and performance of the end product" (Cleland and King, 1983). PMGT is critical to NASA projects which are typically large, complex, involve many different NASA centers and contractors, and are also constrained by time and dollars. Many NASA projects are also unique, and therefore, project managers do not have similar past projects to use as models. Although similar projects may have been performed in the past, the projects are not usually being repeated on a production basis. Therefore, NASA projects, unlike other agencies, can be more difficult to plan and manage effectively.

Many aspects of the PMGT process have evolved into a well-defined methodology, particularly with the development of standard presentation techniques such as PERT, CPM, and Gantt charts (described in Section 2.1). The use of these various techniques assists in the PMGT process because they require the project manager to specify the project goals, develop a project plan, make decisions about each component's requirements, track the implementation of the plan, and evaluate the impact of any deviations from the plan.

The planning and scheduling components of the PMGT process, project management software support, and time estimation are described below.

2.1 PROJECT MANAGEMENT PROCESS

The project management process consists of the following components:

- **Planning** — high-level definition of the project scope, required activities, and resource requirements.

- **Scheduling** — definition of project activities, their relationship to one another, and determination of their resource requirements (e.g., time, cost, personnel, etc.).

- **Monitoring and controlling** — tracking project progress, identifying deviations from the schedule, and determining corrective actions.

Based upon our interviews with NASA personnel, it was determined that the focus of the Phase I effort should be directed towards the planning and scheduling aspects of project management. These topics are discussed in more detail below.

2.1.1 Planning

For the purposes of this report, the term "project" refers to a set of related activities leading to the accomplishment of a goal. Project planning is the process of:

1. Defining the scope of the project, determining project constraints and budget parameters.

2. Identifying the project activities and in what sequence they should be performed.

3. Defining key milestones and deadlines.

4. Estimating time duration, resources, and costs for each project activity.

5. Creating an overall representation of the plan.

The planning process starts with a set of objectives and specifications. These criteria are used to subdivide the project into the smaller steps, or tasks, necessary to complete a project. Each task occurs within a defined time frame and has a starting and ending point. The starting point can be a specific date or be dependent upon the completion date of another task or tasks. Key points of time are specified as milestones which again may be specific dates or be based upon the completion date of a task or tasks. Each task can be further reduced to subtasks and the process repeated as the project manager gradually develops a more precise plan about the specific work to be performed.

A critical part of the planning process is estimating the time required to perform each project activity. The project manager generally develops the time estimates intuitively based upon past experience with similar projects. The time estimation problem will be described in more detail in Section 2.2.

The project plan is frequently displayed as a Gantt chart (as illustrated in Figure 2-1). The Gantt chart is a horizontal bar chart with time represented as a calendar on the horizontal axis and project activities displayed on the vertical axis. Each horizontal bar typically shows the scheduled start and finish time as well as the duration of each major project activity. Numerous variants of the standard Gantt chart have been developed which additionally display such items as project milestones, percent complete, float time, etc. Milestones are used to identify major project goals and usually do not have any associated time duration. They are depicted on the Gantt chart using a special symbol (e.g., a triangle) to identify a single date. Gantt charts are useful for viewing the overall project schedule. However, they are not useful for showing the relationships between activities, determining the critical path, or identifying slack time for non-critical activities.

2.1.2 <u>Scheduling</u>

The project plan is then combined with task dependencies and resource availability in order to develop a project schedule. The processes used to develop the schedule will vary, depending on the type of scheduling required. For example, one may wish to "schedule" the use of a conference room, a truck route, or the development of a Space Station. These scheduling concepts are not directly interchangeable. Basically, there are five types of scheduling:

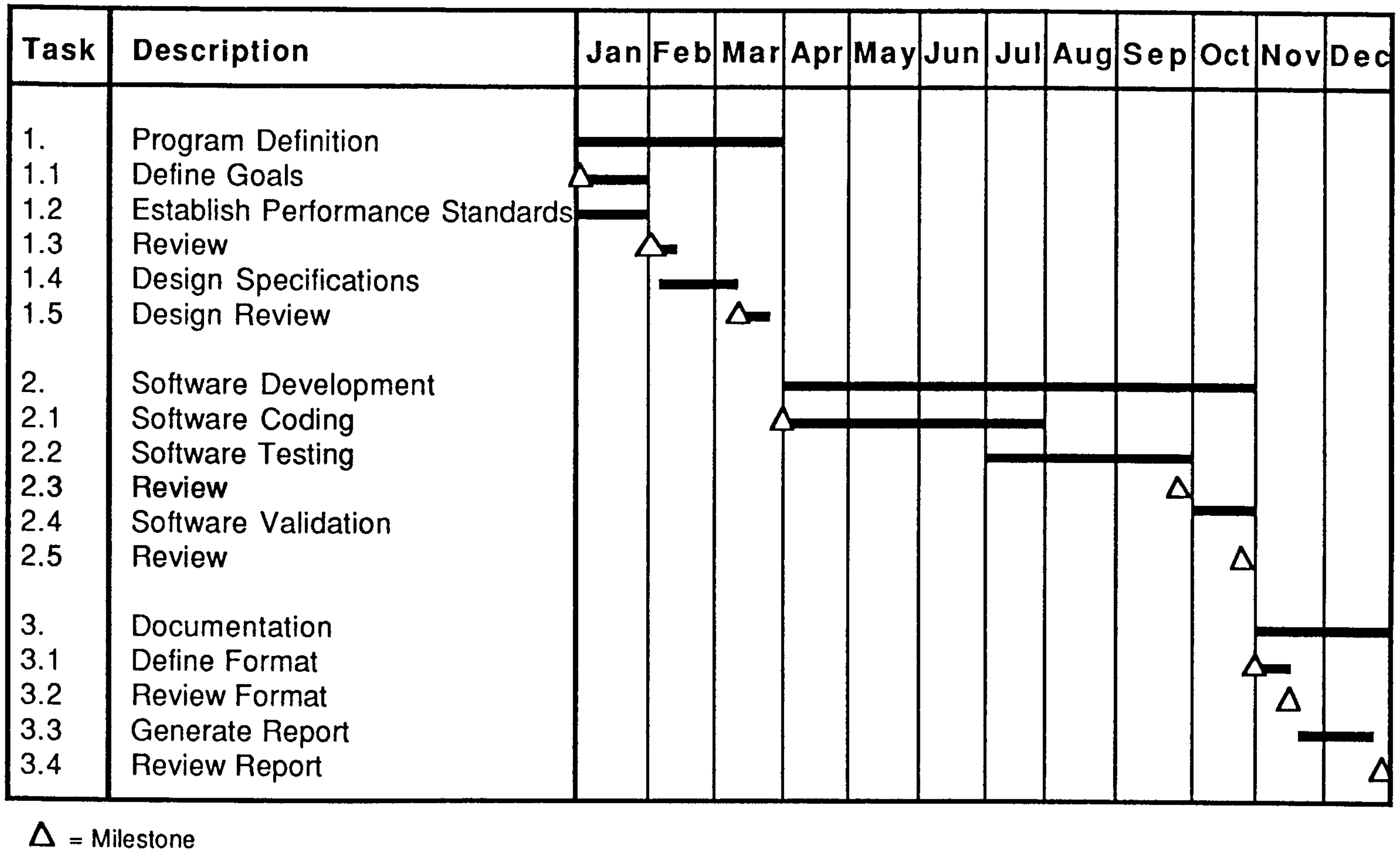


Figure 2-1. Sample Gantt Bar Chart

- **Job-shop** scheduling deals with the problem of deciding the order of items to be processed (as in, for example, a manufacturing environment). The problem is to optimize the flow through the system based on certain constraints such as cost, time, etc.

- **Resource allocation** scheduling attempts to spread limited resources among consuming entities using some rules or heuristics (as in the scheduling of the use of a conference room, assigning labs, or assigning employees to projects).

- **Assembly line balancing** seeks to optimize throughput of an assembly line process based on certain precedence relations and rules. This differs from job-shop scheduling in that it deals with optimizing a repetitious process.

- **Routing** attempts to optimize the itinerary of a single operator as it flows through multiple "checkpoints." This is the well-known "travelling salesman" problem. Given that a salesman must pass through X number of towns, what route should be taken to minimize time on the road?

- **Project scheduling** is the process of breaking a project into its component parts and then defining the interrelationships between the components in such a way that a time schedule can be produced.

Although each of these types of scheduling problems may be applicable for NASA project management, this effort will focus on the last type—project scheduling. Based upon interviews with NASA personnel, project scheduling was identified as the most critical and relevant project management task associated with large-scale development projects such as the Space Station. Therefore, the remainder of this section will address this problem.

Managing large projects involves detailed planning, scheduling, and organization of the project's numerous tasks and subtasks. To aid in this process, **network analysis** techniques have been developed and enhanced since the late 1950's. Two of the methods have reached prominence — PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method) — though the two are often treated as synonymous. For the purposes of this report, these methods will be referred to as PERT-type systems. PERT systems are based upon network diagrams that depict the relationships between project activities. In order to develop a plan using PERT-type systems, the project manager defines each activity for the project, its duration, project milestones, time and resource allocations, and time constraints. Each activity is shown as a node on the network (as illustrated in Figure 2-2). For each node, various time parameters are specified, including start and finish dates. In addition, the dependencies between each activity are also specified (i.e., which activity must be completed before another one can start). The linkage between the activity nodes is also tagged as to whether it is a critical or non-critical pathway. Critical paths indicate the activities that must be performed in successive steps if the project is to be completed on time; non-critical paths represent activities that can occur in parallel. Information such as project milestones, time and resource allocations, and time constraints may also be specified, depending upon the particular variant on the PERT approach being utilized.

PERT-type systems aid in the scheduling and control of large projects by creating a model of the project that can then be adjusted and enhanced in a "what-if" fashion, allowing the project manager to test ideas and to evaluate the results before any decision is made. PERT-type systems reveal "bottlenecks" in the project, thus allowing the project manager to redistribute the resources or, if possible, to correct the problem.

The main concept in PERT-type systems is the **precedence relationship.** This graphically depicts the relationship of tasks and subtasks of projects in such a way that an ordering of tasks can be defined. This ordering then can be used to create a schedule.

Once the order of the tasks has been defined, time estimates are attached to each of the activities, thus creating a schedule. According to Hillier and Lieberman (1974), three times are typically defined for each event:

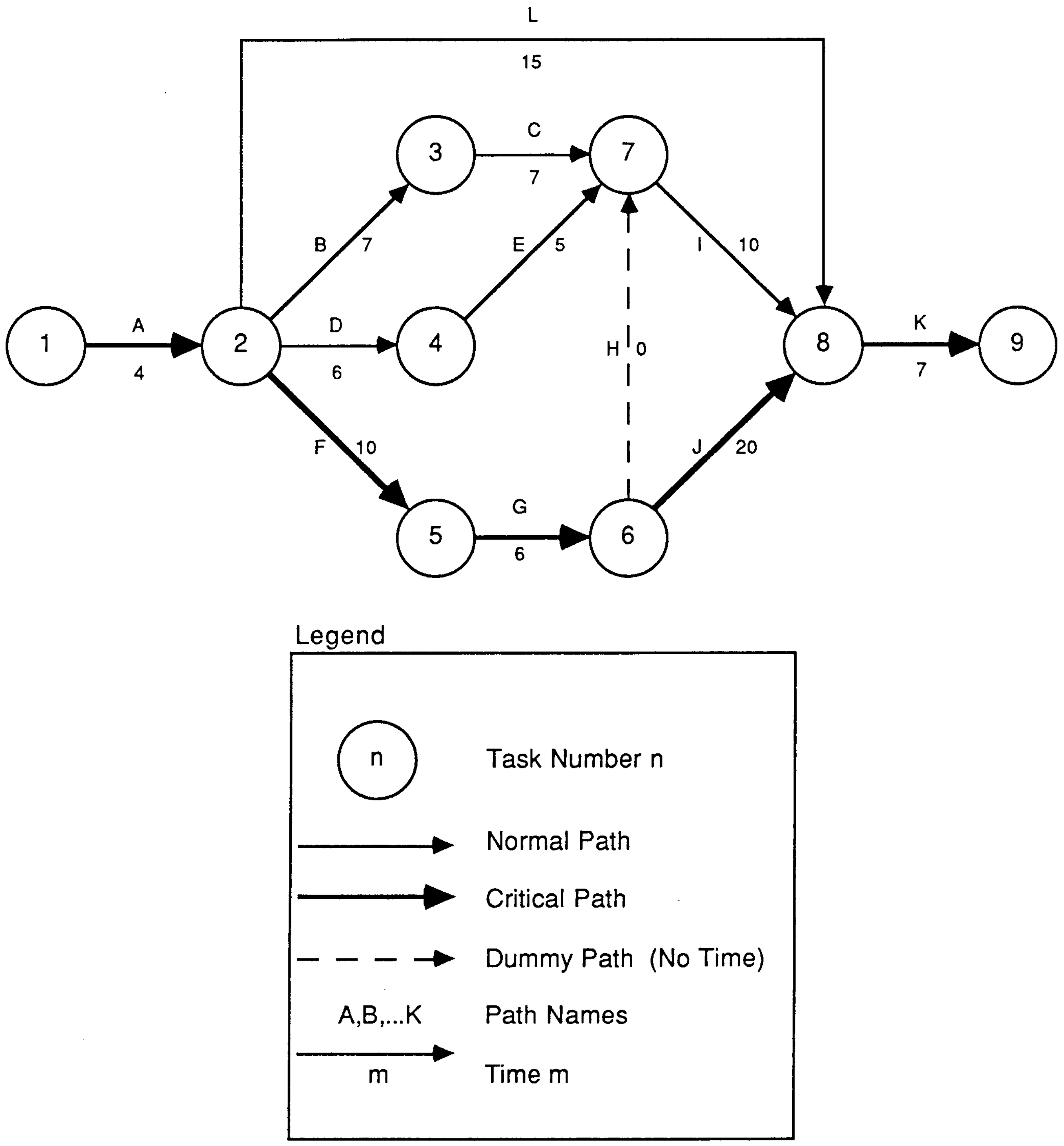


Figure 2-2. Sample PERT-Type Chart

- **Earliest time** — the (estimated) time at which the event will occur if the preceding activities are started as early as possible.

- **Latest time** — the (estimated) last time at which the event can occur without delaying the completion of the project beyond its earliest time.

- **Slack time** — the difference between its latest and its earliest time.

The word **estimated** appears in two of the definitions above (and in the third by reference), which is one of the major problems with PERT-type scheduling methods. Each event time is based on the estimate of the times for each process that precedes it. The question is: "How does the PM make these estimates?" The original PERT method attempted to deal with this problem via the "three-estimate approach" which required the user to make three estimates — most likely, pessimistic, and optimistic — and included a formula to calculate the time for the activity from these additional estimates. The problem with this method is one of infinite regression: in order to develop a single estimate, **three** estimates are required.

This problem has been recognized for more than twenty years as indicated in the following quotation:

> "There is always room for error when estimates are supplied to describe outcomes. Mental gymnastics are required which defy description simply because they are totally internalized and part of the process of cerebral behavior... It would appear that a mental image must be constructed from prior experiences to be as close a representation of the situation that is being analyzed as can be developed from prior experiences... That is why the range of experience of such an individual is important. Unless his library of experience is sufficiently great, he cannot be expected to call up from memory sufficiently good analogs... Therefore, the ability to adapt, alter, interpolate, and extrapolate supposes that some basic pattern of association can be determined." (Starr, 1964)

As a result of interviews with NASA personnel, it quickly became clear that the problem of estimating time requirements for projects is a major problem. The problem is intensified since many of the NASA projects are unique and on the leading edge of technology — and are, therefore, harder to estimate. Currently, time estimates are derived from the project manager's experience base and intuition. The process is very time-consuming, particularly for new project managers, and requires considerable trial and error and numerous iterations. The time estimation problem will be described in more detail in Section 2.2.

2.2 PROJECT MANAGEMENT SOFTWARE SUPPORT

The application of project management methodologies is in a transition phase from a basically "paper and pencil" manual process to a more "automatic" process performed on a computer. This transition has resulted in numerous changes to the PMGT process. The computer relieves the project manager of many previously manual tasks, but the project manager still must develop the project plans and scheduling based upon his knowledge about the project. The TMIS system will include software to support traditional PMGT functions such as the generation of project charts and monitoring project activities. This software will be geared towards relieving the project manager of many charting and bookkeeping functions.

As a part of the current effort, project management tools which incorporate expert systems capabilities were investigated. A tremendous amount of research has been done in both the expert systems and project management domains, yet few efforts have been made to bridge the two domains. Application of expert systems technology to project management has largely been limited to academic research institutions heavily involved in artificial intelligence and organizations encountering significant project management difficulties themselves. Of the expert systems related to project management, the two receiving the far greatest amount of attention in relevant literature are the Expert Mission Planning and Replanning Scheduling System (EMPRSS), by NASA, and Callisto, an experimental system at Carnegie-Mellon's Robotics Institute. These are discussed below. Another less well-known expert system named INNOVATOR will also be discussed.

2.2.1 Expert Mission Planning and Replanning Scheduling System

The Expert Mission Planning and Replanning Scheduling System (EMPRSS) is a prototype project scheduling expert system being developed jointly by the MITRE Corporation and NASA Kennedy Space Center (Hankins et al., 1985). The system's intended use is as an aid in planning and scheduling the activities required to process a payload to be carried aboard the Space Transportation System (commonly referred to as the Space Shuttle). The system's inputs consist of data concerning the flight and hierarchies of tasks to be accomplished. From this data, schedules can be generated at various levels within the hierarchies of tasks and information concerning critical resource consumption.

EMPRSS seems to be midway through development. The system calculates the appropriate flow for mission payload, schedules start and end times for activities, posts requests for resources as needed to accommodate the derived schedule, attempts to resolve resource bottlenecks, and generates alternative schedules to answer "what-if" questions. However, further work has to be done in developing expert heuristics for scheduling and planning — especially for constructing long-term plans and implementing a dynamic planning and replanning system to facilitate short-term dynamic planning and scheduling.

2.2.2 Callisto

Callisto is an ongoing research project at Carnegie-Mellon's Robotics Institute to examine the application of artificial intelligence to large project management (Fox et al., 1986). This work, supported by Digital Equipment Corporation, has led to the development of several expert systems with varying degrees of sophistication. Currently being developed is a system which combines the Institute's previous efforts in activity, configuration, and resource management with the modeling of negotiations and the trade-offs involved as the conflicting goals of multiple managers arise. The system's input data consists of the hierarchies of tasks to be accomplished along with their appropriate relationships, constraints, and construction specifications. From this data, reports are generated providing a description of how to accomplish the project without violating important constraints.

Several project scheduling expert systems have been developed at the Institute, each one producing a more realistic solution as more real-world conditions are considered. The current system performs a wide variety of functions — such as change of orders management; communication among various "mini-Callisto" (computerized) department managers; multi-level scheduling of activities; and generation of PERT, Gantt, and pie charts — and can consider subtleties such as resource management; weak-constraint violation, and availability of space. Yet most of the Institute's work, while ground-breaking in examining the project management problem, is application-specific to large engineering projects and seems to entail the construction of substantial, experiment-specific data bases. Callisto does provide tremendous foundation work for other developments, but still must be considered in the experimental stage.

2.2.3 INNOVATOR

INNOVATOR is an expert system developed for NASA and the military that deals with the problem of reasoning by analogy for system planning (Silverman and Moustakis, 1987). Since it was primarily a research effort, the scope of this system was limited. However, some useful results were obtained in reference to a paradigm for analogical reasoning systems. The small informational base (40 projects) and its development environment (LOTUS on an IBM XT microcomputer) limited the potential "real-world" applicability of the actual developed system.

2.2.4 COCOMO

A number of systems employing the COnstructive COst MOdel (COCOMO) software cost estimating model have been written, including a system called COCOMO1 (Williamson, 1986). COCOMO1 is a software system which guides the user through a series of questions which must be answered in order for the COCOMO cost model to work. After this has been done, the system calculates an estimate using the model and gives results to the user through a series of reports. One problem with COCOMO1 is that it uses a parametric approach to estimating which has been shown to have questionable accuracy (Kemerer, 1987). COCOMO1 serves as a good model for the development of an interface design and the potential for a time estimation system.

2.2.5 Conclusions

A careful examination of current project management expert systems experiments demonstrates that, although such systems are certainly feasible, they pose certain requirements which are difficult to meet. First, because the most popular methodology to implement such systems seems to perform an exhaustive search for providing the solution with the smallest conflict of resources, the systems require the entry of a tremendous amount of instance-specific data. Data concerning all activities and subactivities must be defined, including information specifying the hierarchical relationships and the efforts and resources that the tasks require. These systems also fail to solve one critical aspect of the problem: the quality of project schedules in current systems is proportional to the quality of data specified for each activity, so good estimates of activities yield good schedules, while bad estimates yield poor schedules. Finally, although these systems do provide for plans and schedules to be corrected by changing the data and rerunning the system, historical examples of similar projects which provided good estimates are not specifically offered to replace the user's guesswork.

2.3 TIME ESTIMATION

A common thread to both the project planning and scheduling process is the requirement for the project manager to develop time estimates. Estimating the time that a project (or task) will take is a complicated, error-prone process. In fact, this section could be subtitled "Murphy's Laws of Project Management." There are several reasons for this:

- **Lack of information** — Project managers can only make estimates based on projects that they have worked on, projects that others have told them about, and historical records of projects. However, many times accurate historical information is not available or, at least, not readily available. Thus, estimates tend to be biased toward projects that the project manager knows the most about, which may or may not be appropriate to the estimation task at hand.

- **Lack of resources** — Producing an accurate project estimate will often constitute a significant investment of resources (i.e., time and money). The project manager may be either unable or unwilling to expend these resources for a project that is only in the initial stages of development or for a project that has not yet been funded.

- **Project complexity** — The complexity associated with various project types has a strong influence on the accuracy of schedule estimates. For example, applied research is harder to estimate than production, basic research is harder to estimate than applied research, etc. NASA projects fall within each of these categories and therefore will have varying needs as far as estimation models are concerned.

- **Parkinson's law** — Parkinson's law ("Work expands to fill up time available for completion") works in a recursive fashion against project estimation as follows: An estimate for a project is made, that project (due to Parkinson's law) overruns; therefore, the next time such a project is undertaken, a greater estimate is given and that project overruns. This, in turn, causes the next similar project to be overestimated, *ad nauseam.*

®

- **Process complexity** — Even if the manager had full and perfect knowledge of all appropriate past projects, the task of actually creating an estimate from this information would be fraught with difficulty. The reason for this is that the questions associated with determining the appropriate analogies, valid statistical inference, and combinatorial complexity would overwhelm all but the most tenacious project estimator.

Because of the problems cited above, most scheduling is done "by the seat of the pants" — largely based on intuition. As the person scheduling the project gains experience, intuition typically gets better and estimates become more accurate. Yet there is a potentially severe penalty in incorrectly estimating project schedules.

One way to relieve some of the potential consequences of bad estimations is a technique known as "sensitivity analysis." This is basically a technique for assigning risk factors to the estimate, with the idea of making "best case" and "worst case" plans. As with the PERT estimation process, however, this technique relies mostly on sophisticated guessing.

Another aspect of the problem is that overestimation and underestimation have an additive, not a complementary, relationship. This means that an overestimate does not "balance out" an underestimate, but instead compounds the costs. Moore and Hendrick (1977) note that schedule slippage almost always occurs in the positive direction (i.e., the project was underestimated) and that the slippage tends to be around 50 percent. This seems to indicate that all estimates should automatically have 50 percent added to them, but this has been shown to be incorrect due to Parkinson's law.

3.0 TECHNICAL APPROACH

Project managers typically base their estimates of the time required for project tasks on somewhat arbitrary decisions. In many cases, there is a general notion that a particular type of task usually takes a certain amount of time. For large-scale projects, the project manager's experience may be limited to one or two similar tasks. In such situations, the estimation process combines the knowledge that a specific task in the past required a certain amount of time with a belief it is similar to the task at hand. Experienced project managers tend to develop intuitive models based on previous experience to be used in developing time estimates. There is currently no systematic way of transcending a single individual's relevant experience. Also, formal procedures for evaluating the relevance of isolated historical cases do not exist. Therefore, the primary objective for the Phase I effort is to define a consistent framework for developing time estimates and to conceptualize the features of such a tool. The direction of the Phase I research is to identify the technologies (e.g., artificial intelligence, expert systems, etc.) that are relevant to the time estimation task and determine what role the relevant technologies should play in the development of an expert system to aid project managers in the management of technical projects.

3.1 CONCEPT DEFINITION

The innovative approach to estimating the time required to complete technical tasks that are part of a large, complex project is to develop an expert system that captures and stores the knowledge of program managers. This system will be referred to as the Integrated NASA System for Intelligent Time Estimation (INSITE).

3.2 INSITE SYSTEM: HIGH-LEVEL DESIGN GOALS

Establishing high-level design goals for INSITE is the first step in ensuring the development of a practical and functional system. These design goals include the development of a user-oriented INSITE system and a system structure flexible enough to meet the future needs and changing scope of NASA programs. Ease of programming and maintenance will also be a major goal of the design. A top-down analysis (as illustrated in Figure 3-1) was performed as part of the high-level design goal process which focused on:

- Defining user requirements,
- Allocating functions between the user and INSITE,
- Developing a time estimation methodology,
- Evaluating user-computer interface options, and
- Defining a software development approach.

The system considerations and research for each of these areas are described in this section.

3.3 USER REQUIREMENTS

The first step in developing any system is to define the skills and knowledge of the intended users. The users of INSITE will most probably be project or program managers who are experts for their particular projects and possess extensive amounts of project-specific knowledge that permit very efficient problem understanding, analysis, and solutions. However, the INSITE users may be computer novices with minimal experience in using computers. In addition, INSITE users may possess varying levels of skills for project management, scheduling, and estimation. Therefore, the user interface must be "user-friendly," require minimal computer knowledge, promote rapid learning of INSITE, and provide the flexibility to solve a wide range of problems. INSITE will require extensive dialogue with the user in order to characterize their problem, which requires sophisticated techniques to guide and direct the user in this process. Once INSITE is an established system, the ranks of more experienced users will grow. Therefore, the needs of the novice must be balanced against the needs of a more experienced user in developing the user-computer interface.

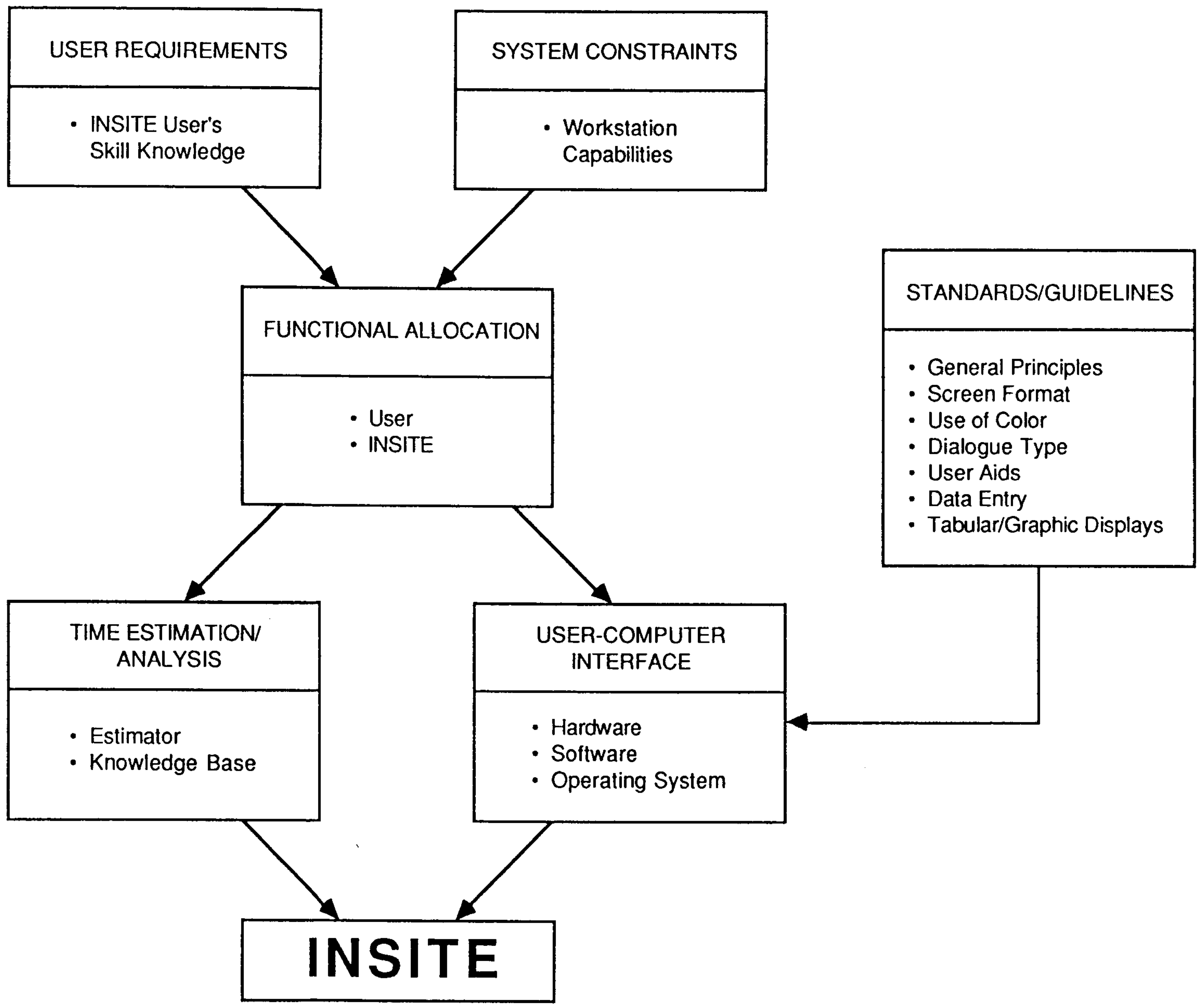


Figure 3-1. INSITE Technical Approach

3.4 ALLOCATION OF FUNCTIONS

The functional allocation step is necessary to determine what system functions should be distributed between the user and INSITE. The INSITE user communicates with INSITE to define and characterize a project, obtain time estimates, and evaluate the results. The INSITE user needs to be provided with a body of systematic guidance on how to develop and validate the appropriate knowledge and system procedures for their particular application. In accordance with the goal of a high-level, user-oriented interface, as many functions as possible will be allocated to INSITE. These include user aids such as menu-driven interface; on-line guidance via a HELP capability; an explanation facility to assist in the interpretation of results; and detailed and easily understood error messages and recovery procedures.

3.5 TIME ESTIMATION METHODOLOGY

The INSITE system will assist project managers in developing more accurate time estimates for their projects. Time estimates are currently developed based upon the project manager's intuition or upon informal comparisons with previous work. In order to determine the methodologies to be considered, characteristics of a "perfect" estimation system were considered.

It would be unreasonable to expect any method of estimation to always return an absolute, correct answer (if this were the case, the system would be calculating instead of estimating). However, the system time estimations should exhibit low deviations when compared to the actual outcome (i.e., it should have a low average deviation and a low maximum deviation). A low maximum deviation is important due to the fact that overestimates and underestimates do not "balance out" (as described in Section 2.3).

Additionally, the system should be **smart** in the following ways:

- It should know that if a task has been done before, this fact should override all other considerations.
- It should know that if an exactly analogous case is known which only differs in degree, that this case is very significant.
- If available information is not very useful, it should indicate that fact to the user but still give an answer.

In short, the system should know what it is doing, not just **how** to do it.

The ideal system should use several approaches to solve the problem and consider each technique when making a determination. Each approach would serve as a guard against one estimate being very biased or totally wrong. Such a system would also be able to explain the determined estimates, including the historical case histories and their relevancy. The system should also allow the user to specify a parameter that should be ignored or given greater weight in this particular case, thus allowing the system to "work with" the user in determining a solution. Section 5 describes several methodologies that are potentially relevant to the development of time estimates, including cost estimation, reasoning by analogy, automatic theorem proving, pattern matching, and fuzzy logic. Section 8 describes how the relevant technologies will be integrated into INSITE.

An additional consideration is the determination of what knowledge is necessary to support an effective time estimation system and how that knowledge should be encoded in such a manner to facilitate its use by the selected INSITE time estimation methodologies. Section 6 presents the technologies that can be utilized for the development of the knowledge base.

3.6 USER INTERFACE APPROACH

Since INSITE will be a complex system, the nature of the INSITE user-computer interface (UCI) will be an important determinant of successful system operation and effective performance. The UCI will be designed to take advantage of the most recent advances in human factors engineering, artificial intelligence, and advanced display technologies. A primary design goal is the development of a system which is maximally flexible and adaptable for new applications and which is highly usable in terms of both operation/execution and analysis of results. The design of the UCI will be considered during every phase of the development of INSITE to ensure that INSITE incorporates high degrees of flexibility, adaptability, consistency, and responsiveness with regard to the way in which the user will interact with all the components of INSITE. It is essential to integrate human factors into the early stages of the design process where it can have the greatest benefit. A good user interface design may cost more in terms of time and money to implement, but it may also result in significant benefits during the system's productive life. Non-productive training time can be reduced; user misunderstandings leading to interpretation errors can be avoided; and user satisfaction can be dramatically increased.

The development of the user interface will be based upon analysis of the following:

- **Interactive dialogue analysis** — establishing dialogue style (e.g., menu, command, graphics, etc.), user response, data entry screen design, on-line help, error message design, and color coding.
- **Input device and techniques analysis** — examining properties of available input devices (e.g., keyboard, mouse, etc.) and their interaction with the dialogue.
- **Output requirements analysis** — examining properties of available output devices and information to be conveyed (e.g., text, graphics).

Another design goal is that the user interface should be consistent across all components of the INSITE system. Other principles that have been identified in the human factors literature as critical to successful user interfaces include the following:

- "Friendly" dialogues and error handling,
- On-line help routines,
- Meaningful feedback provided to avoid confusion,
- Minimal strain on human memory capacity,
- Simplicity rather than complexity, and
- Demands tailored to the user's skill levels.

Wherever possible, the design of the user interface will be in accordance with established human factors guidelines and standards.

Section 7 describes the technologies researched for the development of an effective INSITE computer interface.

3.7 DEVELOPMENT APPROACH

Traditional software development approaches assume that all system requirements can be precisely determined and specified in detail prior to any contextual design, implementation, or operational experience. The design specifications are frozen at some point, and the entire system is based upon these specifications. By contrast, the prototype approach assumes that precise requirements are not always predefinable so the system is developed utilizing a building block approach. A building block approach to system development develops a working foundation of a system quickly in such a manner that it can be gradually expanded one step at a time. The premise of prototyping is that it is easier and quicker to modify and improve a tangible system than to draw up specifications for a system that can handle every conceivable requirement. This is particularly true for expert and knowledge-based systems where an iterative approach is required to establish and refine the knowledge base. Prototyping

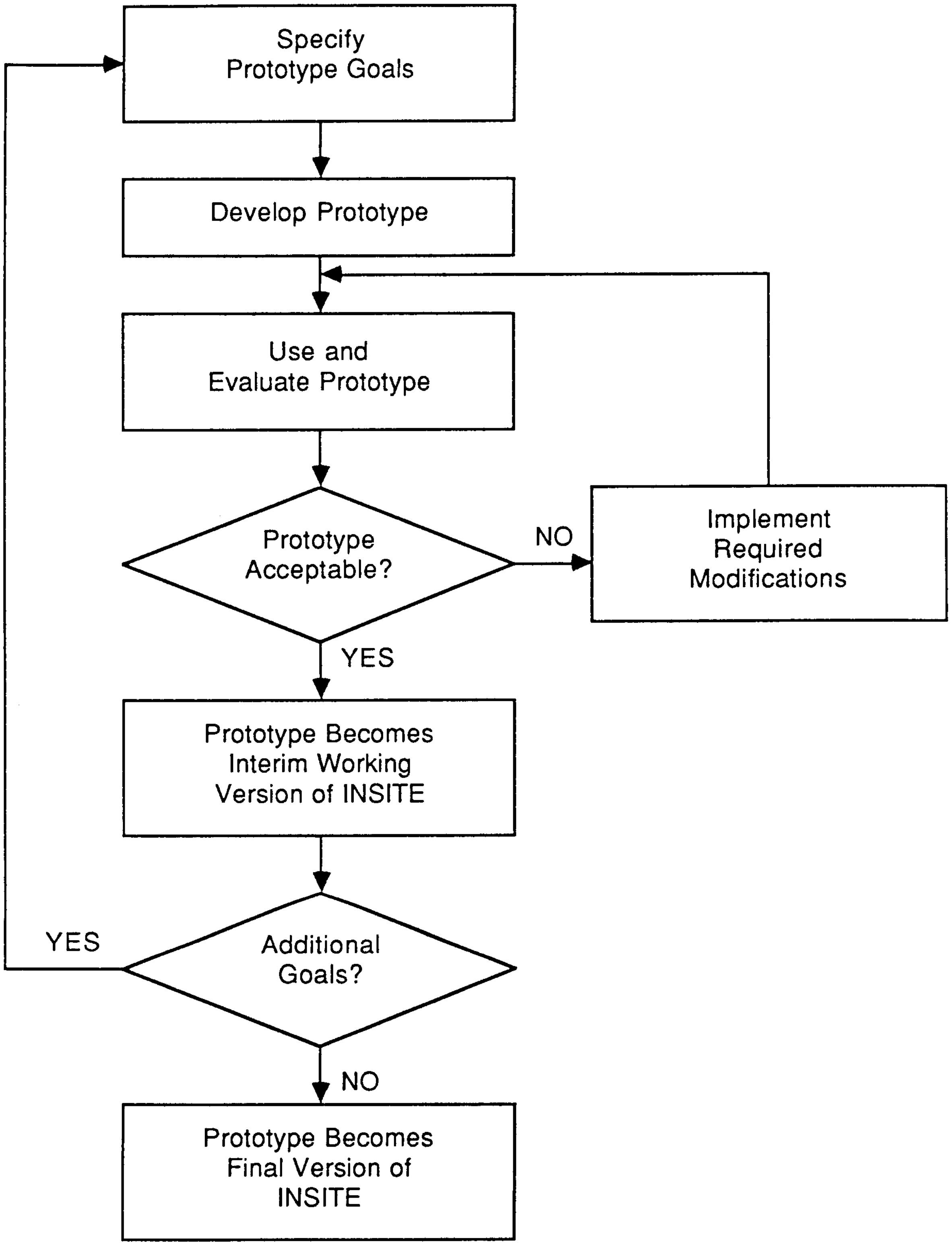

Specify Prototype Goals
Develop Prototype
Use and Evaluate Prototype
Prototype Acceptable?
NO
Implement Required Modifications
YES
Prototype Becomes Interim Working Version of INSITE
YES
Additional Goals?
NO
Prototype Becomes Final Version of INSITE

Figure 3-2. Prototyping Software Development Approach

begins in the analysis phase of system development with a first prototype based on a high-level functional analysis. It does not include every feature the eventual system might include, but at each stage implements the desired goals effectively with minimal development costs. The prototyping approach, illustrated in Figure 3-2, consists of the following steps:

1. **Specify prototype goals** — clearly identify the scope of the prototype and determine how it is to be evaluated.
2. **Develop prototype** — design and implement prototype; determine what functional modules must be developed and how they will be integrated with modules in the current operational version.
3. **Use and evaluate prototype** — demonstrate the prototype to the user in the context of actual applications and elicit feedback from the users in terms of how the prototype meets their needs and requirements.
4. **Implement required modifications** — incorporate any indicated modifications into the prototype and repeat the evaluation process.

When a particular prototype is functioning satisfactorily, it is made available to the user for evaluation to determine if the system development is on the right track and performs as expected and/or required. Users can knowledgeably suggest changes that will improve the system and make it more applicable to their needs. The system developers can then incorporate those changes with a clear understanding of what exactly needs to be changed and how it should look when completed. Each succeeding version of the prototype more accurately reflects the users' requirements and incorporates more of the features of the eventual system. The prototyping process is reiterated until all system goals have been developed and evaluated.

The traditional approach to software development is best suited for systems with simple and static requirements. But for dynamic and complex systems, the best way to develop the system is with a prototyping approach. The user's understanding of a system is an evolutionary process. Changes of meaning and structure of the system reflect the learning process and growth that accompany every application experience. In order to increase the usability of the system, it is necessary to accommodate these changes, not to impede them. An approach that exposes the user to realistic versions of the final application will lead to wide exploration of the application alternatives during the earliest stage of development

Several studies have indicated that a prototype approach significantly improves the probability that a useful system will be developed and that the overall development cycle will be shortened (Mason and Carey, 1983). Additional experimental results suggest that prototyping increases the actual utilization of a system by the user, and systems performance (as measured in terms of user satisfaction with the system and its perceived accuracy, utility, and functionality) was rated higher by users of prototyped systems than by users of systems developed using traditional approaches (Alavi, 1984).

4.0 ESTIMATION AND RELATED TECHNOLOGIES

The primary objective of INSITE is to create time estimates for project managers. The relevant technologies that were thoroughly investigated can be grouped into two categories: algorithmic and heuristic. Algorithmic methods are well-defined, have a "step-by-step" process, and are guaranteed to have a solution. Heuristic approaches are based on general strategies or rules of thumb. The algorithmic and heuristic approaches are discussed in Sections 4.1 and 4.2. Section 4.3 provides a description of expert systems technology.

4.1 ALGORITHMIC APPROACHES

4.1.1 Cost Estimating

The methodologies utilized for cost estimation are analogous to those used for time estimation. Cost estimation procedures are regularly used in construction, defense contracting, software development, and many other areas. Cost estimating can be done in a variety of ways as described in the following sections (Defense Systems Management College, 1983).

4.1.1.1 Bottom-Up Costing. Bottom-up costing is considered an estimation procedure even if, as the name implies, it is a method of summing the costs of components to derive a total cost figure. The cost of the project is defined as the sum of the costs of each component plus an "overhead" or "administrative" cost. The problem with this method is that it, like many similar estimation methods, only postpones estimation. To say that the cost of a project is the sum of its components is definitional. The question becomes: "What are the costs of each component?" Unless one is working on a fixed-price contract basis, this method of estimating is of limited use. However, it does compel the project manager to reduce the project into lower-level components such as a work breakdown structure. This process is useful since any attempt at structuring the project is bound to enhance the PM's knowledge of the situation which, in turn, will aid in the creation of estimates.

4.1.1.2 Comparison. If a PM has knowledge of a previous project that is very similar to the one in question, this can be used as a model for making estimates about the current project. However, there are several questions associated with this process:

- How dissimilar can the projects be and how much dissimilarity can a good estimate tolerate?
- If the dissimilarity is great, how should the estimates be adjusted in order to account for the differences?
- What aspects of the projects should be considered when making the comparison?

These questions have no simple answers and, in fact, are applicable to many of the estimating methodologies discussed in this report. However, the nature of cost estimating is very error-prone due to the lack of definition of the methodology.

4.1.1.3 Statistical Analysis. The problem with the previous two methodologies for cost estimating is that they are too subjective. A method that does not rely solely on the PM's particular knowledge and biases is preferred. In this regard, the use of statistical analysis seems appropriate. In an ideal situation, a PM could simply look up the cost of a project in a book, based on some criteria. For example, the building/construction trades often refer to a structural cost handbook which lists average costs for different types of structures by region. This method works for the building trades since there are numerous examples from which to draw the "average" cost, and the processes are well-defined so that meaningful comparisons can be made. In the case of more complex projects, however, problems of data collection and problems of definition of appropriate criteria of comparison combine to make this method of estimation unsuitable.

4.1.1.4 Parametric Analysis. Mathematical modeling is an estimation technique that is less subjective. This approach describes the cost of a project via equations based on a set of input parameters or comparison factors. The appropriate comparison factors are developed based upon the emphasis of the particular model and adjusted based upon the "goodness" or "badness" of the estimate for a particular project.

The Programmed Review of Information for Costing and Evaluation (PRICE) model was developed by RCA for the Defense Systems Management College (1983). This system is an example of parametric analysis. Its usefulness is based on the fact that it does not incorporate a single model, but several. As with any such system, it needs calibration for the particular user's project. PRICE also includes a list of the factors that are the most important for a variety of project types.

The major problem with this method is that often the model is not suited for the project at hand. In fact, in a study of four of the most popular software cost estimation models, an average error rate was found to be 460% (Kemerer, 1987).

The trade-off for parametric analysis techniques tends to be between generality and accuracy. As a model becomes less general, its usefulness begins to decline since more work is needed to determine which model to use as well as identifying the appropriate factors. In the worst case, the estimation method degrades to the "bottom-up" approach described in 4.1.1.1.

4.1.1.5 Conclusions. The problems associated with cost estimation are very similar to the problems of time estimation. However, the cost estimation techniques currently available seem to have minimal utility for NASA's time estimation problem due to the fact that they (a) are not applicable (particularly the statistical method), (b) require excessive "up-front" time (particularly the bottom-up method), or (c) provide inaccurate results. Thus, while current cost estimation methodologies can be very instructive, it remains an area where further research and development is needed.

4.1.2 Statistical Estimation

Statistical estimation is used in virtually every scientific and business related enterprise. Its usefulness is almost axiomatic; virtually no one will question a justification by **valid** statistical analysis. It will be seen in Section 7 that statistical analysis will be integral to the design of INSITE; more detail as to the approach will be given at that point.

4.1.3 Pattern Recognition and Pattern Matching

Pattern matching and recognition, which combine a statistical approach with computer science and mathematics, is central to the study of robotics, signal processing, and AI. Pattern matching determines the degree of similarity between a set of data points describing a particular situation and a previously constructed data set. Given a set of data points, these techniques determine:

a. If there are patterns which define classes of data points; and

b. Given a new data point, if it can be placed in any existing class from which information about the point can be inferred.

The relevance to project scheduling and time estimation is clear: if information about historical projects can be obtained and patterns detected, perhaps an inference can be generated regarding current projects.

Using the pattern recognition approach to estimate project costs, a set of n measurements, called features, is defined to represent a common set. Each of the n features is connected to an n-dimensional feature space, with each task having a feature vector. The main problem is to find a match between the feature vector of some known source (or group of sources known as a cluster) and the feature vector of the source in question. For our purposes, the problem is to find a cluster into which the project fits and then, based on information about that cluster, derive a time estimate.

Cohen and Feigenbaum (1982) have classified the algorithms for pattern matching into four categories, which are each discussed below.

4.1.3.1 Statistical Pattern Matching. There are numerous algorithms for this process. One basic approach is the **nearest neighbor classification** approach, which seeks to find the nearest data point to the point in question and determine if the new point belongs to the cluster of the point nearest to it. That is:

If there exists a k,l such that $| X_{k,l} - X | < | X_{i,j} - X |$ for all i,j

where $X_{x,y}$ is the yth sample in the x class and

X is the unknown vector

then X is assigned to class k.

This is a parsimonious approach, but is error-prone due to the fact that clusters can overlap. This technique yields unreliable results since proximity to a point does not necessarily denote homogeneity. A variation to this scheme would be to sample a set of closest points and assign the class of the unknown to be the most common class within that set. This is somewhat better, but still suffers from the same false assumption.

An effective variant of this method uses the distance to the center of the cluster to define the class. This means that any point whose center is nearest is in the class. This is an appropriate strategy if the clusters are well-defined and symmetric, but not very accurate otherwise.

It is also possible to define the boundaries between classes through the use of probability density models. These models take into account not only proximity to the center of a class cluster, but also the relative density of the points at that location as compared to the density of the next nearest class cluster. The boundaries are established to minimize the cost of error, not just the occurrence of it.

None of these models allow the analysis of clusters with complex shapes or classes with multiple clusters. Research on more complex analysis techniques is ongoing, though further discussion of these techniques is beyond the scope of this paper.

Michalski and Stepp (1983) indicate that one of the major limitations of these types of approaches is that the a priori classes defined for the data often lack any simple conceptual interpretations. They cite the reason for this as the implicit assumption that all of the attributes of the data sources are of equal relevance, whereas in reality, some of the properties will have overriding importance to the cohesion of the group, while others are of little importance. They suggest an approach (conjunctive conceptual clustering) in which the attribute selection is performed simultaneously with the formation of clusters.

4.1.3.2 Parametric Pattern Matching. In parametric pattern matching, the list of possible parameter vectors (rule space) is searched by "hill-climbing" or gradient descent in order to find the class that minimizes the error between the model and the unknown. Typically, the following linear functional is used:

$$y = wx = \sum_i w_i x_i$$

where x is the feature vector and

w is a weight vector, the elements of which are the unknown parameters.

What is needed, then, is to find the hyperplane that separates the classes. If none exists, then either a more complex relationship can be found or a hyperplane which results in low error percentages can be used. The algorithms used to find this hyperplane are explained in Cohen and Feigenbaum (1982).

4.1.3.3 Pattern Matching Automata. The Pattern Matching Automata method seeks to find "a finite-state automaton whose behavior imitates that of the unknown system" (Klaus and Horn, 1986). In this approach, the system is defined from an initial state and the transitions from one state to another are described. One method describes the transitions via a probability matrix, while another uses fuzzy set membership as the criterion (fuzzy set theory is described in more detail in Section 4.2.5). Pattern Matching Automata have not as yet produced much in the way of practical application.

4.1.3.4 Structural Pattern Matching. Structural Pattern Matching borrows from formal language theory in that it attempts to define a grammar for a particular class. An unknown is parsed (checked against a defined grammar) and is placed in the appropriate class. Stochastic grammars use statistical analysis in an attempt to accommodate ambiguous or ungrammatical patterns. Few practical structural pattern matching algorithms have been proposed, but work is continuing and may yet yield some useful results.

4.1.3.5 Conclusions. A problem associated with all of the pattern matching techniques is the selection of the feature measurements. Klaus and Horn (1986) contend that "no amount of sophistication in the decision algorithm can make up for a poor selection of features." Thus, the selection of the comparison criteria is crucial to the success of the classification. Since pattern matching capabilities will play a significant role in the estimator portion of INSITE, a large portion of the design time will be devoted to the definition of the relevant features of projects which will be used for the comparisons.

4.2 HEURISTIC APPROACHES

4.2.1 Reasoning by Analogy

In general, time estimates can be made in two different ways based on two different types of knowledge. First, an estimate can be based on an understanding of the factors which determine project time requirements. If this understanding is complete and detailed, direct calculations can be made to produce an estimate. If domain knowledge is less complete and at a more general level, a series of heuristics/rules may be applied to generate an estimate. Second, an estimate can be made based on past experience. That is, a project may be estimated by reference to some other similar project (or projects) which has already been completed and whose time requirements are therefore known. This second approach depends on the ability to identify completed projects which are somehow "analogous" to the project in question.

4.2.1.1 Analogy in Everyday Experience. Analogical reasoning has long been recognized as a major component in human thought. It is integral to our problem solving strategies from the hard sciences to the most trivial everyday activity (Sternberg, 1977):

> "Reasoning by analogy is pervasive in everyday experience. We reason analogically whenever we make a decision about something new in our experience by drawing a parallel to something old. When we buy a new goldfish because we liked our old one, or when we listen to a friend's advice because it was correct once before, we are reasoning analogically."

It is the fact that analogy is a central technique for dealing with new situations that has caused a number of disciplines, from psychology to the history of science, to focus on analogical reasoning. The most substantial body of work has been produced by psychologists who have, for some time, recognized the relation between analogical reasoning and intelligence. Anyone who has ever taken an intelligence test is familiar with the classic formulation "A is to B as C is to D", where the subject is presented with a multiple-choice option for "D". It is assumed that the ability to deal with new situations (as seen in a person's ability to manipulate analogies) is central to the notion of intelligence. Even 60 years ago, Spearman (1927) claimed that "it is certain that [analogy] tests — if properly made and used — have correlations with all that are known to contain *g* [Spearman's general factor of intelligence]."

Although the concern is as old as the interest in analogy itself, recent work in psychology has focused on understanding analogy as a process. Some researchers continue to extend a traditional approach by attempting to reduce analogical reasoning to a series of low-level (now information processing) components. Sternberg's five-component theory (1977) can be used to exemplify this strategy. The components themselves are best described in the context of an analogy problem: Washington is to 1 as Lincoln is to (10,5)? The answer to the problem is "5" because Washington appears on the one-dollar bill and Lincoln appears on the five-dollar bill. According to Sternberg's model, the steps taken to arrive at this answer are the following:

1. **Encoding**: The subject begins the solution with attribute identification for all terms in the analogy. Attributes and values which may be relevant for analogy solution are retrieved from memory. For Washington, these might include "first president", "portrait on dollar bill", and "revolutionary war hero". For Lincoln, these might include "sixteenth president", "portrait on five-dollar bill", and "Civil War hero".

2. **Inferring**: The subject attempts to relate attributes of the first and second terms in the analogy. Two attributes of Washington, "first president" and "portrait on dollar bill", are found to relate to attributes of the digit 1 (namely, ordinal position and amount).

3. **Mapping**: The subject attempts to relate, by means of retrieved attributes, the first and third terms in the analogy. The subject notes that Washington and Lincoln were both presidents, were both war heros, and are both depicted on currency.

4. **Application:** The subject attempts to construct a relation between the third term and each of the candidate solutions, based on the relations already inferred. No relation between Lincoln and the number 10 can be found on this basis. The fact that Lincoln appears on the five-dollar bill allows the subject to construct a relation between Lincoln and 5 which is analogous to one of those inferred between Washington and 1, and therefore arrive at a unique solution.

5. **Response**: The subject responds with "5", having completed the analogy problem.

Although inferring, mapping, and application may appear to be the most complex components in this model, the selection of "relevant" attributes is a major problem. Aspects of human thought which are critical to our understanding of analogical reasoning are simply swept aside by explanations such as the following (Sternberg, 1977): "Potentially relevant attributes are those that experience has indicated are useful in relating one concept to other concepts." The idea that the structure of memory itself may play a major role in analogical reasoning has only recently emerged. Rumelhart and Abrahamson (1973) believe that portions of semantic memory can be represented as a multidimensional space. In this case, analogies are treated as similarity judgments and can be assessed directly since distance is a straightforward computation within the context of a multidimensional Euclidean space. Other models which are capable of extending or replacing the process-component approach will undoubtedly emerge as our understanding of brain functioning improves.

Most of the analogies dealt with in intelligence testing are straightforward, usually based on relationships which are simple, well-understood, and easily verbalized. In technology and science, however, analogies are often complex and imperfect. Whether an analogy is made to a past problem solution or a past theory, the analogy may be relatively poor, yet provide a framework for continuing efforts (Oppenheimer, 1956):

> "At each point the first scientists have tried to make a theory like the earlier theories, light, like sound, as a material wave; matter waves like light waves, like a real, physical wave; and in each case it has been found one had to widen the framework a little, and find the disanalogy which enables one to preserve what was right about the analogy."

This subtle kind of process is also seen in everyday problem solving and remains a potentially fertile area for continuing research on reasoning by analogy.

4.2.1.2 Automating Natural Reasoning. Since its inception, artificial intelligence has been motivated by an interest in emulating "intelligent" human behavior. In the context of a desire to automate natural reasoning, it is not surprising that AI research focused on the human ability to solve analogy test problems. It is hard to imagine a more striking demonstration of machine intelligence than a computer taking an IQ test. Twenty years ago, computer programs already existed which could accomplish this task. ARGUS could solve verbal analogies (Reitman, 1965), and ANALOGY could solve geometric analogies (Evans, 1968).

ANALOGY is a classic example of the describe and match paradigm. The relation between any two geometric objects can be described as a series of geometric operations (e.g., rotation, reflection, scaling) applied to the first object in order to obtain the second. To answer the question "A is to B as C is to (D_1, D_2)", the program determines which geometric transformations are necessary to get from A to B, from C to D_1, and from C to D_2. The correct solution is that solution which requires the same set of transformations as, or the most transformations in common with, the set required to go from A to B.

A reliance on matching underlies most AI approaches to analogy developed in recent years, including complex schemes such as Winston's learning by analogy (1984). The ARCHES system (Chouraqui, 1985) provides an alternative approach to the automation of analogical reasoning. Figure 4-1 presents the analogical paradigm in the ARCHES system, following the "A is to B as C is to D" formula. In general, to discover a new D, the system first assesses the similarity between A and C. For example, the knowledge base may include the fact that "the earth is round" and that "the earth is habitable by humans". Since the moon is also round, by analogy we might determine that the moon is habitable by humans. The inference is invalid for two reasons: 1) the inference is based on the resemblance of very incomplete descriptions of the earth and moon, but equally, because 2) there is no dependency relation between "the earth is round" and "the earth is habitable by humans". For this reason, the paradigm (as shown in Figure 4-1) stresses that analogies are valid when a dependency relation exists between "A and B" and "C and D" as well as when a similarity exists between A and C. In fact, the "degree of probability of conclusion D is as high as the relation of dependency is strong." (Chouraqui, 1985). The dependency criteria, which "only experts are in a position to evaluate and justify in proportion to their scholarship," are the responsibility of the user of the system who establishes objects and attributes through a series of dependency graphs (Chouraqui, 1985):

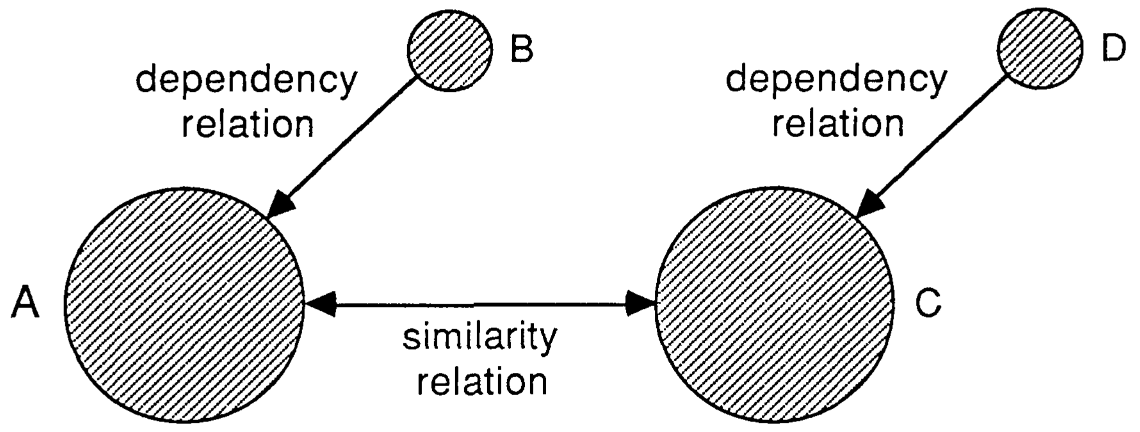


Figure 4-1. Representation of Analogical Paradigm in the ARCHES System

"This mode of using the rule of analogical inference — *filtered by the dependency graphs* — contributes to the production from a set of logically true (i.e., valid) structures, sets of structures whose truth depends on interpretations defined by these dependency graphs."

4.2.1.3 Conclusions. Research by psychologists into the nature of analogical reasoning has given rise to a number of related procedural models. At best, however, these models are only adequate for the analysis of simple cases of reasoning by analogy. More complex processes remain poorly understood. AI approaches have also been successful in dealing with straightforward analogies. The underlying matching strategies have even been extended to more complex situations. As in other expert system approaches, automated analogical reasoning is dependent on human expertise which has been captured in system structures. In the case of analogies, it is the dependency relations which must be defined by the human expert.

4.2.2 Symbolic Logic

Symbolic logic (as opposed to syllogistic or classical logic) has been one of the main methods of attacking the problem of intelligent systems. Hundreds of years of development in this field had taken place before computers came into existence and, in fact, the field of computer science owes most of its existence to a logician named George Boole (father of Boolean logic). Logic itself seeks to be a method of discriminating between valid and invalid arguments; symbolic logic systems are designed to provide precise, formal standards of validity (Haack, 1978). As such, AI researchers have used these formalized techniques of reasoning within computer systems to attempt to mimic human thought.

Though there have been a number of symbolic logic systems (SLS) developed throughout the years, there are some consistent threads which connect them all. Each must begin with a symbology, or a system of symbolizing the primitive (base) concepts with which the system will be dealing. The main primitive in many logic systems is the statement (also: sentence or proposition), which is any sufficiently informational sequence such that that sequence must be true or false. For example, "Adam is three" is a statement, but "Adam" and "three" are not. In most cases, an SLS will symbolize the statement "Adam is three" as "A". Another statement, such as "Adam is terrible", would be symbolized as "T".

Next, a procedure to relate the statements to one another is defined through the use of statement connectives. The following statement connectives are frequently used:

•	And,
v	Or,
~	Not,
$\supset$	Material implication, and
$\leftrightarrow$	Material equivalence.

Thus, the statement "If Adam is three then he is terrible" would be symbolized "A $\supset$ T"; the statement "If Adam is three and Adam is terrible then Adam is typical" would be symbolized "(A • T) $\supset$ U" (where "U" stands for "Adam is typical"); and the statement "Adam is not typical if and only if Adam is not three or Adam is not terrible" would be denoted as "$\sim U \leftrightarrow (\sim A \vee \sim T)$". This system allows for arbitrarily complex statements to be symbolized. The symbology used is designed to be "English-like" and is therefore more verbose than is necessary. In order to symbolize any two-valued (i.e., TRUE or FALSE) logical statement, only the two operators ~ and • (or, alternatively: ~ and v, or ~ and $\supset$) is required.[1] This fact is important for the discussion of automatic theorem proving (see Section 4.2.3).

The symbols described up to this point make up what is known as **sentential logic**; that is, the logic of sentences (statements). Sentential logic yields some interesting results, but it is very limited in its representational capabilities. Predicate logic expands upon the base of sentential logic by adding the ability to assign properties to individuals or classes of individuals using the following conventions[2] :

Capital letters (A-Z) denote *property constants*;

Small letters (a-t) denote *individual constants*; and

Small letters (u-z) denote *individual variables*.

A property constant is any descriptive aspect of an individual (such as tall, smart, etc.). The phrase "Bob is smart" could be symbolized as Sb, with "S" standing for "is smart" and "b" standing for "Bob". This greatly enhances the capacity to represent a concept.

Another type of phrase is the generalization which can be expressed using quantifiers in either of the following forms:

Existential quantification:

$\exists(x)\ (\theta)$ "There exists an x such that proposition θ is true."

[1]Actually there are two operators that are sufficient in themselves: "|" (which may be understood as "not both _ and _) and "Ø" (which may be understood as "neither _ nor _").
[2]Uses symbology derived from: Kahane, Logic and Philosophy, pp. 38-56.

Universal quantification:

$\forall(x)\ (\theta)$ "For all x, proposition θ is true."

The above provide SLS with the capability to represent a wide range of propositions. But the symbolization of the statements is only half of the concept. A facility to make valid inferences from these symbols must be included in any SLS. This is handled by a set of valid deduction rules which can be mechanically applied to a proposition or set of propositions with the guarantee of deriving a true statement (given that the premises are true). By applying these rules appropriately, one can prove that given a certain set of premises α, a conclusion β follows necessarily.

As a tool to enhance human thought processes, SLS have been proven through years of use. However, there are problems using SLS as a model for AI systems design; three such problems are discussed below.

4.2.2.1 Translation from Natural Language into Symbolic Logic Notation. The translation of an English sentence into a logical proposition is a non-trivial process. In fact, Copi (1954) states "It must be emphasized that there are *no mechanical rules* (italics added) for translating statements from English into our logical notation." And, although Otto (1978) differs on this issue, it is obvious from his work that it is not an easy task to define the rules for this translation. As computers are machines, mechanical rules are necessary for them to perform a task.

What constitutes a "good" symbolization of a proposition is also a question. It is a necessary, but not a sufficient, condition that a symbolization be correct in order for a symbolization to be "good." The symbolization must also be sufficiently detailed for the proof at hand; otherwise, some necessary information may get "lost in the translation." For example, the sentence "If Adam is three and Adam is terrible then Adam is typical" could be symbolized as "X", just as correctly as the symbolization "$(A \bullet T) \supset U$". The difference between these two translations is in the level of detail.

4.2.2.2 No Algorithm for Symbolic Logic Proofs. The SLS described above has no algorithmic (step-by-step) process for doing proofs. Choosing the rule to apply (and to what to apply it) at any step in a proof is the main challenge in doing proofs. In any non-trivial proof, the number of possible alternative strategies is astronomical. Thus, even if the premises are adequately symbolized (see above), doing the proof is still a problem for a computer, though there have been attempts (Otto, 1978). However, automatic theorem proving (as described in Section 4.2.3) attempts to work around this problem.

4.2.2.3 No New Information. One severe limitation to the entire deductive inference methodology is that symbolic manipulation of propositions yields no new information. That is, what is obtained is only a conclusion as to whether a series of statements are logically consistent — nothing new is discovered. This is inherent in the system; if guaranteed correction is required, then insight is lost.

4.2.2.4 Conclusions. SLS are a useful tool for humans, but are less applicable (except by analogy) to machine intelligence. The theoretical basis for symbolic logic systems is, however, one of the major underpinnings of AI and computer science in general. The next section delves into a methodology for applying SLS to AI.

4.2.3 Automatic Theorem Proving

Though it has been around since the early 1960's, automatic theorem proving (ATP) is a subject that has recently gained interest due to its applicability to AI in general and expert systems specifically. It is basically an approach to making mechanical derivations from premises to conclusions based on the principles of logic. It is a subfield of symbolic logic (as described in 4.2.2) in that it has the same theoretical basis, but differs in its orientation. Whereas the systems of symbolic logic described previously were created for use by humans, the ATP techniques are particularly suited to computers due to their computational complexity.

4.2.3.1 Clausal Form. In order for mechanical processes to be performed on SLS propositions, they must be converted into a standard form, called clausal form, in which no operators are used except ~ and v (as previously stated in Subsection 4.2.2, these two operators are sufficient to describe any two-valued logical statement). There is a mechanical procedure for converting any logical statement into clausal form (Clocksin and Mellish, 1984), which involves manipulating the logical symbols in a repetitious fashion. First, all conditionals and biconditionals are replaced by their logical equivalents as shown below:

$(A \supset B)$ is replaced by $(\sim A \vee B)$

$(A \leftrightarrow B)$ is replaced by $((\sim A \vee B) \bullet (\sim B \vee A))$

Next, all negations are manipulated so that they are attached to atomic formulae (anything which is not a complex statement), using the following rules (Cohen and Feigenbaum, 1982):

$\sim(\sim A)$ is replaced by A

$\sim(A \bullet B)$ is replaced by $\sim A \vee \sim B$

$\sim(A \vee B)$ is replaced by $\sim A \bullet \sim B$

$\sim\forall x\ A(x)$ is replaced by $\exists x \sim A(x)$

$\sim\exists x\ A(x)$ is replaced by $\forall x \sim A(x)$.

Next, all existentially quantified variables are replaced with constants unless it is within the scope of a universal quantifier (in which case, it is replaced by a function on the universally quantified variable). For example:

$\forall x\ \exists y\ P(x,y)$ is rewritten as $\forall x\ P(x, f(x))$.

Universal quantifiers are unnecessary after this, since we can assume that any variable that was not existentially quantified is universally quantified. Thus, the example above would be written as:

P(x,f(x)).

The last step is to convert to clausal form by applying DeMorgan's laws:

~(A • B) is replaced by ~A v ~B

~(A v B) is replaced by ~A • ~B

Once this process is complete, the next step in ATP is resolution.

4.2.3.2 Resolution. The resolution algorithm is the basis of ATP. It is the process of trying to prove a proposition by showing that the proposition's negation yields a contradiction. The basic reasoning method is:

(~P v Q)

(~Q v R)

(~P v R)

For example, given:

Either Paul is not smart or the Queen is and

Either the Queen is not smart or Ralph is.

Then it is true that:

Either Paul is not smart or Ralph is.

A necessary function of any ATP is unification, which is the process of finding substitution instances for the variables in a statement in clausal form. This allows the ATP to find values to "plug into" the propositions in order to make deductions based on the information given by them.

There are algorithms both for unification and resolution. The major problem with resolution is that the search space (i.e., the number of combinations that must be examined in order to resolve a theorem) grows exponentially with the number of premises in the problem description. This means that even relatively small problems can take an unreasonable amount of time to solve. Some heuristic strategies have been developed which cut the processing time, but they have the drawback of not being complete (that is, they are not guaranteed to give an answer).

4.2.3.3 Data Base Management and Deduction. One class of problem in which ATP's are particularly suited is in the area of data base access. If all information in a data base were complete (that is, every possible property and relation were spelled out explicitly), then deductions would not be necessary. However, such a situation is technologically unfeasible as well as nearly theoretically impossible. The number of relations between the elements of even a small set of data can be almost infinite and therefore unmanageable. A better way to deal with this situation is to define relations between classes of data objects, then perform deductions on them. For example, given the following data base:

Project	Type
1	α
2	β
3	γ
4	α
5	β
6	β
7	α
8	α

and given that we know that all projects of type β and γ are of type θ, we might want to ask: "What are all of the projects of type θ?" In order to answer this, a field "meta-type" could be added to the data base as follows:

way to deal with this situation is to define relations between classes of data objects, then perform deductions on them. For example, given the following data base:

Project	Type
1	α
2	β
3	γ
4	α
5	β
6	β
7	α
8	α

and given that we know that all projects of type β and γ are of type θ, we might want to ask: "What are all of the projects of type θ?" In order to answer this, a field "meta-type" could be added to the data base as follows:

Project	Type	Meta-type
1	α	ω
2	β	θ
3	γ	θ
4	α	ω
5	β	θ
6	β	θ
7	α	ω
8	α	ω

This scheme would require more storage space. Alternatively, we could build the following rules into a data base retrieval mechanism:

Project	Type	Meta-type
1	α	ω
2	β	θ
3	γ	θ
4	α	ω
5	β	θ
6	β	θ
7	α	ω
8	α	ω

This scheme would require more storage space. Alternatively, we could build the following rules into a data base retrieval mechanism:

All projects of type α are of type ω;

All projects of type β are of type θ; and

All projects of type γ are of type θ.

The above would provide for the same operation without using the extra storage space. It could be argued that this is simply a trade-off between storage and processing time (since the rule processing would take some time). However, this argument falls apart if a new piece of knowledge is added: all projects of type α or β are of type ρ. With the first method, a new field would be added, called "meta-type 2," and each of these types, along with the project information, would be stored. With the ATP method, only one more rule is added. This classification of projects could continue *ad infinitum*, and the first strategy would quickly become infeasible. This, then, is a class of problem to which ATP techniques are quite well-suited.

4.2.3.4 Conclusions. Automatic theorem proving is a very useful technique for solving certain types of problems. However, it is not a panacea since the type of reasoning used by this approach is very stilted in that it allows for the representation of only a small class of problems. Also, the use of an ATP in an expert system requires a large up-front investment in knowledge engineering to be effective since the rules must be very carefully coded to be of any use.

4.2.4 Probabilistic Reasoning

Symbolic logic systems are most often designed to deal with statements about which there is no uncertainty (Rauch, 1984). For example,

(1) If evidence E is true, then hypothesis H is true.

Frequently, however, information is not "black or white"; there is usually a measure of uncertainty associated with it. In order to denote uncertainty, the following formats are used:

(2) If evidence E is true, then hypothesis H is true with probability P_1 (confirming evidence) or

(3) If evidence E is not true, then hypothesis H is true with probability P_0 (disconfirming evidence).

This switch from certainty to uncertainty allows the acquisition of new knowledge (Strawson, 1966) and is the basis of scientific reasoning. Thus, techniques for valid inference with uncertainty have been developed. By combining probabilities with symbolic logic, valid deductions can be made, along with a calculation of a degree of certainty. The degree of certainty indicates how likely it is that the consequent of the deduction is true. The mechanics of these processes are discussed below.

4.2.4.1 Probability of an Hypothesis. Using the P_1 and P_0 probabilities described in (2) and (3) above, it is possible to calculate the probability of a certain hypothesis (P_H) to be:

$$P_H = P_1 P_E + P_0(1-P_E)$$

where P_E is the probability of the evidence being true.

The standard deviation of the error in the probability estimates can then be used as a measure of confidence (σ_H) in the calculation. It can be derived as follows:

$$\sigma_H = (P_1 - P_0)\,\sigma_E$$

where σ_E is the standard deviation in the error estimate of the probability of the evidence.

4.2.4.2 Types of Probabilities. The probabilities P_0 and P_1 can be either stated or derived. Stated probabilities are values obtained by experiment or definition (i.e., it is a given). Derived probabilities are based upon the manipulation of two or more probabilities. There are two main schools of thought as to the origin of these values. The classical theory of probability states that probabilities are measures of rational expectation (Kahane, 1973). For example, when throwing a die, most people know that there is a 1 in 6 chance of rolling a "three." This is an example of an a priori probability — the value is determined before any dice are thrown. The frequentist theory determines probabilities based on experimental evidence. A frequentist would determine the probabilities of throwing a "three" by throwing a die numerous times and dividing the number of occurrences of "three" by the number of times the die was thrown. Sometimes a stated probability will be based on an individual's estimate (in which case, there may or may not be any rationalization at all).

As stated above, derived probability is determined by some manipulation of one or more other probabilities, which may themselves be either stated or derived. At some point, of course, some stated probabilities must be given. The rules for the combination of probabilities are discussed below.

4.2.4.3 Dependence Relations. Central to the discussion of probabilities is the notion of dependence. Two events A and B are independent if the probability that A happens has no effect on the probability that B happens and vice versa. Two events that are not independent are either maximally dependent or minimally dependent. The notions of maximal and minimal dependence do not, as the names might seem to indicate, differ as to the strength of the dependence between the two events. An event A is minimally dependent on an event B when A is maximally dependent on **not** B (that is, there is a negative correlation between A and B). The strength of the correlation remains the same.

4.2.4.4 Probabilities and Logic. There are formal methods for calculating the probability of an event E, where event E is an established dependency between events A and B (E = A • B). The dependency relationship can be stated as:

Independent events:

	B TRUE	B NOT TRUE
A TRUE	$P_A P_B$	$P_A(1-P_B)$
A NOT TRUE	$(1-P_A)P_B$	$(1-P_A)(1-P_B)$

Maximum dependence:

	B TRUE	B NOT TRUE
A TRUE	$MAX(P_A,P_B)$	$P_A-MAX(P_A,P_B)$
A NOT TRUE	$P_B-MAX(P_A,P_B)$	$MAX(1-P_A,1-P_B)$

Minimum dependence:

	B TRUE	B NOT TRUE
A TRUE	$P_A-MAX(P_A,1-P_B)$	$MAX(P_A,1-P_B)$
A NOT TRUE	$MAX(1-P_A,P_B)$	$1-P_A-MAX(1-P_A,P_B)$

When the events are independent, it is easy to calculate a standard deviation which can be used as a measure of confidence in the result. When the events are not independent, a standard deviation can be estimated to arbitrarily close tolerances (Rauch, 1984).

4.2.4.5 Probability Trees and Bayes Theorem. Another approach to calculating probabilities is by using a probability tree, which is illustrated in Figure 4-2. This method allows the calculation of all possible probabilities of multiple tests using the multiplication theorem of probability (Lipschutz, 1965):

$$P(A \bullet B) = P(A)\, P(B \mid A)$$

which states that the probability of A and B is equal to the probability of A times the probability of B given that A has occurred.

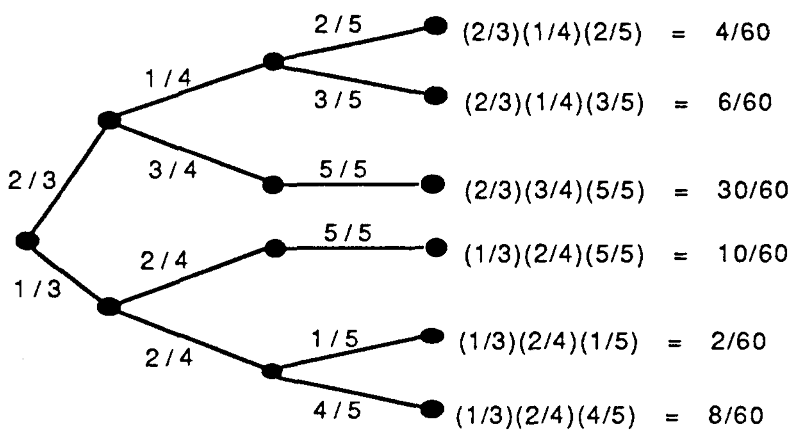


Figure 4-2. Probability Tree

Bayes Theorem provides a technique to calculate the probability of A given that B has occurred as follows (Charniak and McDermott, 1985):

$$P(A|B) = \frac{P(A)\, P(B|A)}{P(B)}$$

An expert system named Prospector, which is a system for predicting the location of ore deposits, uses a Bayesian Inference Network. The Bayesian Inference Network is based on a principle of intermediate states (Charniak and McDermott, 1985). These states define a network of valid inferences, each with associated probabilities, and allows situation modeling; the system searches for patterns of situations in which ore deposits occur.

4.2.4.6 Conclusions. The concept of probability enhances the use of inference systems in that it allows valid conclusions to be drawn in the absence of complete information. At some level, however, the system still relies on a somewhat hazy notion — that of rational belief. The origin of the stated probabilities is still an open question. Nevertheless, despite theoretical questions, probabilities often provide a "commonsense" approach to problem solving. The ability to manipulate probabilities is central to most rule-based expert systems in use today.

4.2.5 Fuzzy Logic and Fuzzy Set Theory

At the heart of many AI problems is the basic fact that human problem solving often occurs under conditions of vagueness or uncertainty. The uncertainty may be in reference to the information used or the methods employed, or both. For example, given the following rule:

(1) If a patient's temperature is high, then the patient has a fever.

The following questions could be asked:

What temperature is "high" and

Is rule (1) always true?

In a traditional computer program, "high" would be defined as crossing some threshold (e.g., 100°F) and the rule would be used as if it were always true. Thus, if the patient's normal temperature were 96°F and his current temperature were 99°F, the program would not diagnose the patient as having a fever. Likewise, if the patient had just been removed from a vat of near-boiling water, the fact that his temperature was high would not necessarily indicate that he was suffering from a fever.

Traditional computer science is built upon Boolean logic, which is, in turn, based on set theory. One of the basic axioms of set theory is the concept of the excluded middle, which states that the following is always true:

$A \vee \sim A$

This translates into the fact that an element cannot be both **in** a set and **not in** a set at the same time. Though this seems self-evident enough, it is also a very limited way of looking at things. Consider, for example, the set of "expensive things." Most people, it is assumed, would agree that a Rolls Royce belongs in this set and that a pack of bubble gum does not. But what of a Ford Escort? Since it is a car, it is an expensive item. Yet, in relation to other cars, it is not very expensive. And what of a pack of gum that costs $1.50? Suddenly, this gum is in the same category as a Rolls Royce! There are two principles at work here: vagueness and open textureness. A word is vague if it is used differently at different times, but a word has an open texture if the lines of differentiation are not defined (and also not definable). Vagueness can be overcome by stricter definition; open textureness cannot.

It is easy to dismiss this discussion as a problem of semantics, yet there is a fundamental concept here: human thought processes are not usually as "cut-and-dried" as traditional logic and set theory would have it. For computers to begin to more closely emulate human thought processes, appropriate fuzzy representation schemes would have to be devised. This was the impetus for the creation of fuzzy set theory and its corollary, fuzzy logic.

4.2.5.1 Fuzzy Thinking. Fuzzy set theory has, as its base concept, sets without sharp borders. Subsets of these sets have elements, each of which has an associated "degree of membership." The degrees of membership are defined as a number between zero and one. For example:

tall:height -> [0,1]

which might be defined as (Negoita, 1985):

TALL

5'0"	0.00
5'4"	0.08
5'8"	0.32
6'0"	0.50
6'4"	0.82
6'8"	0.98
7'0"	1.00

The normal types of relations are defined for these fuzzy sets: complementation, conjunction, disjunction. By combining fuzzy relations, new relations can be defined (such as "middle sized" from "not tall and not short"). All of the normal properties (i.e., communitivity, associativity, DeMorgan's laws) apply to the fuzzy set operations. Thus, an entire "fuzzy logic" is built.

A compositional inference technique uses inferences made from sets of membership values and known relations between sets (Negoita, 1985). This technique permits the development of inferences between known and unknown values based on a correspondence between the membership values of the sets in question. For example, given the membership value of the subset TALL of the set HEIGHT, the inference could be made that the subject had a certain WEIGHT (within some certainty range) if there was a known relation between the set HEIGHT and the set WEIGHT.

4.2.5.2 Conclusions. Fuzzy set theory and fuzzy logic allow for a closer correspondence between the way humans think and the requirements of a computer system. In this way, human thought processes can be more readily captured in an AI or expert system. For example, it would be beneficial for the user to be able to use such words as "large," "expensive," etc., and have these concepts correctly interpreted by the system.

4.3 EXPERT SYSTEMS

An expert system codifies the specialized problem solving expertise of an authority (and, in some cases, many authorities) to assist in solving complex problems in narrow domains. Expertise in a specific domain may generally be described as knowledge about the domain, the problems involving the domain, and the methods and approaches to solving the problems. Particularly important types of expert knowledge are the heuristics or "rules of thumb" which allow an expert to effectively evolve solutions.

The terms "expert system" and "knowledge-based system" are often used interchangeably to refer to AI-based systems that capture expertise in problem domains. In this report, a **knowledge-based system** is considered to be a system consisting of two **separated** components:

- A **knowledge base** representing the heuristics, facts, judgments, and experience about a selected problem domain.
- An **inference processor** that interprets the contents of the knowledge base to infer conclusions toward a solution of the problem.

The inference processor (or inference engine) very often incorporates an automatic theorem prover and probabilistic inference capabilities as discussed earlier in this section. A full discussion of knowledge encoding is in Section 5. The separation of the knowledge from the inferential mechanism permits more flexible development of the application and more closely follows how humans deal with complex problem domains.

An **expert system** is considered to be a type of knowledge-based system in which:

- The knowledge base represents **human** expertise in the domain;
- The inference engine mimics **human** expert reasoning processes; and
- The system's inferential processes are available to the user for review.

Traditionally, expert systems are generated by a "knowledge engineer" who extensively questions an expert in a particular field to determine information and know-how about a selected topic and translates the expert's knowledge into a knowledge base. As expert systems have become more available and sophisticated, it has become clear that this knowledge base construction process is both the heart of and the main bottleneck to building an expert system. Thus, less time is now available for the creation of the inference mechanisms themselves. This has led to the increased use of expert system shells, which are tools for creating expert systems that eliminate the need for "reinventing the wheel" (or, in this case, the inference engine).

4.3.1 Expert System Shells

The use of a good inference engine is critical to the success of building an expert system. An expert system shell provides numerous program modules to handle a variety of tasks, thus freeing the knowledge engineer from having to deal with these details. Programming the functions an expert system shell performs is generally considered an overwhelming effort: the modules must not only exist, but must also perform extremely efficiently. The more complete a shell is, the easier the knowledge engineer's job will be, since he will have more tools at his disposal. The more efficient a shell is, the more likely the system will actually be used, since expertise will be offered more quickly.

The modules included in an expert system shell vary, but typically include:

- A rule editor to create the structures the knowledge engineer is using to represent the information he has gathered during the knowledge acquisition phase,

- A consistency checker to determine if the structures entered are logically consistent (a structure might be redundant or conflict with others),

- An inference engine to perform the actual reasoning of the system to move from problem to solution by applying the knowledge base,

- A data base manager to store the knowledge base, and
- A user interface to make the shell easy to use.

Each of these modules may vary tremendously from shell to shell in sophistication. For example, a system's rule editor might require the structures to be entered as programming-type statements in some computer language or as abstractly as a series of pictures representing conditions which, when satisfied, derive a result. Similarly, an inference engine might search for solutions by simply exhaustively applying rules or could incorporate such features as using multiple paths of reasoning and certainty factors. The functional sophistication of a shell determines the level of effort a knowledge engineer must expend to make an expert system perform in a given manner.

Besides considering the effort a knowledge engineer must make, the modules' efficiency also must be considered. Especially important in designing an expert system is the inference engine's capabilities since, as the number of rules in a system increases linearly, the time needed to reach a solution can increase geometrically. This exponential explosion of solution time has proven the downfall of many otherwise excellent expert systems; no user can afford to let a system spend hours making a decision needed in seconds. Finally, memory efficiency is also a significant consideration; knowledge bases tend to be very large, and when hardware constraints become significant, programming can become a nightmare.

Expert system shells vary greatly from implementation to implementation, but the better ones provide numerous, efficiently programmed functions essential to creating an expert system. Shells provide functions which are not normally feasible to program due to the large developmental costs involved. Therefore, one heuristic concerning building expert systems is to choose a shell wisely since the shell's quality is often an important factor in determining the system's successful development and use.

5.0 KNOWLEDGE REPRESENTATION TECHNOLOGIES

There are three types of information necessary to the successful operation of INSITE — historical project data, project knowledge, and user knowledge. The historical project data base will utilize standard data base management system technology (probably of the relational type). The technology driving this aspect of HISTORIAN is well-understood and readily available in "off-the-shelf" software packages. A knowledge base is distinct from a data base in that a knowledge base incorporates not just data, but the relationships among the data, the relationships among the types of data, and further, the relationships among the relationships among the data and types of data. Information for project and user knowledge will require the capabilities of a knowledge base, and this section presents information about relevant knowledge relation schemes and how this knowledge can be acquired.

5.1 KNOWLEDGE REPRESENTATION TECHNIQUES

The representation of knowledge is the heart of any expert system and refers to employment of an explicit symbolic representation of the information in its domain of concern. One of the hypotheses of AI is that knowledge is representation: that is, "knowing" consists in large part in representing facts about the world symbolically (Brachman, 1985). Selection of the techniques used to represent the project and user knowledge is perhaps the most important decision about the INSITE system; this decision determines both the kinds of problems that can be solved and the methods of solving these problems. Research in AI knowledge-based systems has identified several major types of knowledge representation (KR) techniques which are summarized below.

5.1.1 Procedural Representation

In a procedural representation, relevant knowledge is embodied in "procedures" (i.e., subroutines that can **do** specific things in well-specified situations). Procedural representations have the advantage of capturing a large amount of knowledge, including heuristics, economically. On the other hand, the "big picture" may be lost and the underlying knowledge is not easily retrievable or modifiable.

5.1.2 Semantic Nets

A semantic net represents objects, concepts, and events as nodes in a network (and the interrelationships between them as links) and is based upon the concept formulated by Woods (1975). The nodes can be linked either through memberships in class (the "is-a" concept) or as a subproperty of another node (the "has-a" concept). Semantic nets provide a very flexible structure, and additional nodes and links can be added at any point. The semantic net is most useful to represent relationships between objects. The "is-a" and "has-a" relationships permit an inheritance capability such that any characteristics altered in a node can be automatically carried through another node. However, a major problem with the semantic net for knowledge representation is that a given net may have several interpretations and a given meaning may be reflected in several different nets.

5.1.3 Frames and Scripts

Frames and scripts, based upon the ideas of Minsky (1975), are techniques to represent the sequence of events and properties that **typically** occur in a given situation in an organized fashion. A **frame** is a knowledge representation structure in which new data is interpreted in terms of previous experience. A frame has **slots** to represent all the attributes of interest. Slots can contain factual, descriptive, or procedural information. Slots can also represent another frame so that an inheritance hierarchy is established (i.e., lower-level frames inherit knowledge about the associated higher-level frames). Most frame techniques for knowledge representation also incorporate provisions for generic frames to be established for various object types. Frames have the capability to represent a great complexity of information and are a very active current research area. Some of the important unresolved frame-related issues are control issues (such as determining the appropriateness of a given frame and selecting a second frame if the first is not appropriate).

Scripts may be viewed as a special class of frames in that scripts embody a large amount of previous knowledge in "typical" situation representations. Scripts are specifically designed to represent knowledge about **events**; a normal or default sequence is represented as well as possible exceptions or errors. As with frames, there are procedural attachments with scripts.

5.1.4 Production Rules

Production rule knowledge representations are based upon conditional statements that specify an action that is to occur under a certain set of enabling conditions. The rules are generally stated as two-part statements in the form: "If this premise is true, then perform this action or make this conclusion." Each rule is evaluated, and when the current condition matches the premise stated in the IF rule (i.e., the condition is TRUE), then the indicated action is performed. Such rules permit explanation of system conclusions as a sequence of logical steps. Production rule techniques are most useful for presenting procedural knowledge (i.e., methods for accomplishing goals). Frequently, production rule techniques also incorporate forward and backward chaining rules and a pattern matching capability. Forward chaining matches rules against facts to formulate new facts; backward chaining attempts to prove a new rule by determining what facts are required. Pattern matching utilizes complex algorithms to formulate decisions based upon the best match to current conditions. Rule-based knowledge representation techniques have become dominant in current expert systems development. However, production rules become unwieldy and difficult to manage as the number of rules increase since rules can be added that conflict with previous specified rules.

5.1.5 Conclusions

On the surface, it might appear that the "is-a" type of representation found in semantic nets would be ideal for representing historical project data. In this case, however, we are interested in encoding nothing more complex than sets of project attribute values, and typical data base record structures are sufficient. Semantic nets and frames are capable of representing very complex relationships and carry a correspondingly large overhead for storing and processing very simple types of information. Project knowledge (the source of and relations between attributes) and user knowledge (profiles) are a very different matter. Here the amounts of data may be small, but the relationships may be complex. For example, on software development projects, the time needed to complete the project may be increased due to manager inexperience. At the same time, this effect may be offset by experienced programmers working in a highly productive development environment. Knowledge of this type could, for some purposes, be represented in semantic nets or similar structures. INSITE, however, requires that project knowledge and user knowledge be used to drive computer-human interaction and single-analog estimation. A production rule system is the appropriate form of knowledge representation in this case. Such a system can easily capture both the relationships between data items and the relationships between data and required actions. Unlike procedural representation, production rule representation is also consistent with a rapid prototyping approach to development since a rule base can be continuously modified in an efficient manner.

5.2 KNOWLEDGE ACQUISITION

Knowledge acquisition applies the psychological and statistical aspects of information gathering to artificial intelligence, thereby providing an essential element of any expert systems endeavor. Knowledge acquisition entails the elicitation of consistent, comprehensive information from expert sources organized in a concise fashion for inclusion into a knowledge base (KB). The importance of knowledge acquisition to a system providing expertise on project planning/scheduling cannot be overstated. Once information concerning historical project estimates has been examined and the planning/scheduling process well-understood, a body of knowledge can be compiled from which expertise can be offered.

Several types of information are needed to populate the knowledge base, principally **structures** and **heuristics**. Structures of various types, including rules, frames, and propositions, provide the elementary facts needed to solve specific pieces of a problem. Heuristics, on the other hand, are the more intuitive or "rules-of-thumb" considerations that an expert uses in solving a problem. Heuristics provide general guidelines which focus the problem solver, whether human or machine, on subsets of the domain in search of a solution. Both types of information must be included where any significant application of expert systems, including project scheduling, is undertaken. Information structures comprise the individual facts that are steps along the course to a solution, while heuristics provide the control to keep the path straight. The proper use of these two types of information is of paramount importance in expert systems development. Although structures provide solutions, heuristics ensure a solution within a reasonable amount of time.

The following are descriptions of the most common methods used to acquire domain knowledge to construct an initial KB. Accompanying each technique is a critique of its strengths and weaknesses. It is important to realize that often a combination of methodologies is used in the knowledge acquisition process to ascertain information in a top-down (with a progressive expansion of detail) or bottom-up (with an increasingly generalized view) fashion as needed to facilitate the knowledge engineer's (KE) comprehension of the domain. Multiple techniques can also be employed to implement a more intelligent system since some methods are geared towards creating an initial knowledge base and others towards implementing a learning system whose knowledge base grows through use. The relationship of these processes to the knowledge engineering problem as a whole is rather complex (as illustrated in Figure 5-1).

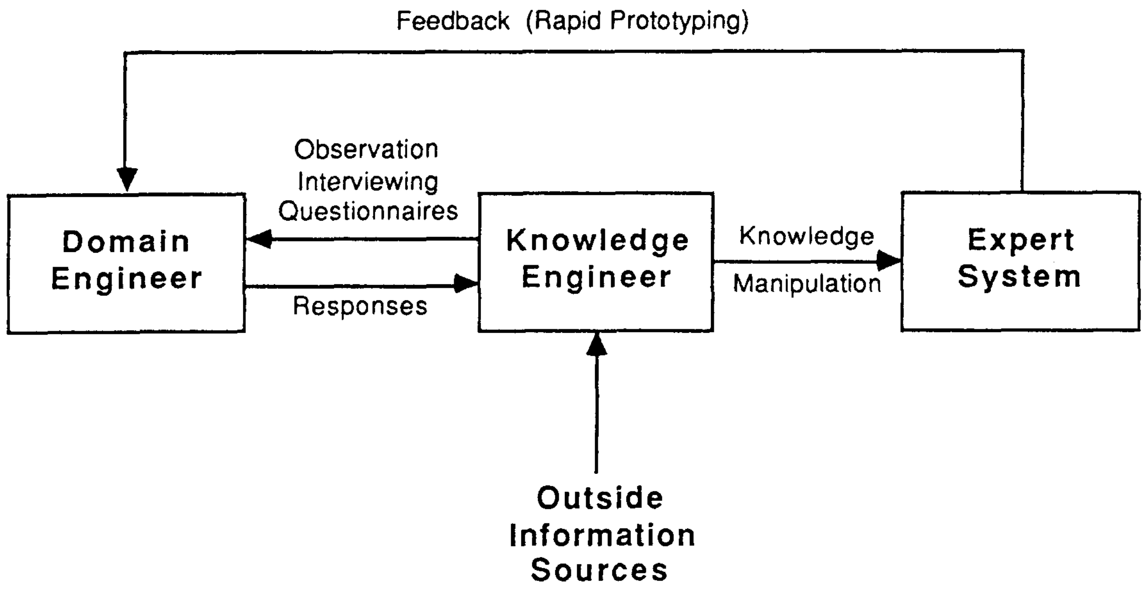


Figure 5-1. Knowledge Acquisition Process

5.2.1 Initial Knowledge Base Construction

5.2.1.1 Interviewing. Interviewing is the obtaining of information via the KE's direct discussions with one or more domain experts. This technique is commonly used to determine the objectives, methods, and structures which should be implemented in the proposed system. This method allows the KE to directly query the domain expert (DE) and to immediately qualify any ambiguities which arise. Also, subtleties in solutions visible only via direct interactions with an expert are noticed, providing insight unavailable by many other methods; interviewing facilitates discovering nuances such as exceptions to rules. However, since the KE dynamically interacts with the domain expert, the course of the session is unknown. Therefore, interviewing is an inexact science implemented using general guidelines, not formal procedures, to follow. Interviewing requires the continued cooperation of one or more experts over an extended period of time. Thus, a variety of organizational problems arise, including obtaining experts, determining the validity of differing expert opinions, and scheduling experts with limited amounts of time. Since the process is very human-dependent, a variety of psychological issues must be accounted for, including maintaining the experts' cooperation and interest, allaying their fears of technology and job insecurity, and dealing with interpersonal communication barriers.

Related to interviewing is another knowledge acquisition technique — observation. Observation is a technique whereby the KE observes the domain expert solve actual problems. It has many of the same advantages and disadvantages of interviewing. Observation does provide an exceptionally useful tool to understand the specific structures and procedures used to attack an actual problem. This method also is used to evoke information about aspects of the problem assumed as trivial.

These two techniques are often combined with rapid prototyping to provide successively more sophisticated implementations of an expert system. By having the experts use each prototype system and provide feedback, advice is generated with increasing fidelity in each version. However, the method requires the ongoing efforts of a KE. Once the knowledge base has been set in place, information structures cannot be altered without the programming of a knowledge engineer. Other techniques must therefore be used to implement a true learning system.

5.2.1.2 Questionnaires. The technique of questionnaires involves the construction and distribution of inquiries to domain experts and the encoding of responses by the KE. This method generally has the benefit of requiring little time and inconvenience on the part of domain experts. Another advantage is that it provides the statistical information which can be used to derive general rules, avoiding the problem of creating a rule for one instance. Unfortunately, however, this method offers organizational barriers to being implemented iteratively for feedback purposes and cannot be used to incrementally augment the knowledge base after the system has been developed. Further, the psychological and statistical considerations involved in constructing and interpreting questionnaires are numerous. Research on complex analysis techniques has been done and is ongoing, but further discussion on the topic is beyond the scope of this report.

5.2.1.3 External Sources. A variety of external sources can be used to compile information for a knowledge base. The simplest form consists of the outside information sources the domain expert uses in the normal course of work (such as drawings, tables, and handbooks). These types of sources offer expertise without domain experts and so are a convenient method to obtain information. Yet, for the same reason, these sources offer a fixed amount of information; if the KE is unsure of some presented concept, another data gathering technique would be necessary for further clarification. Consequently, external sources are most often used to acquire only basic information and for explanation purposes within the system.

Another type of external sources is the use of examples or case studies. The use of examples is accomplished by providing data concerning historical instances of the process to the system. The data, in turn, is manipulated in some fashion by the expert system to extract and derive useful information. An example of using this technique might be an expert system modeling a courtroom judge (Schank, 1984). Ongoing entries of case histories of criminal behavior could continually refine the system's structures to more accurately reflect factors such as repeated criminal behavior, tendencies to violence, and abuse of parole privileges. Using examples entails many complicated considerations such as establishing proper metrics, determining how data is to be manipulated, and managing large quantities of information (since, generally, as more cases are entered into a system, the more accurate the information becomes). Further, since a system using this technique is highly dependent upon the user's evaluation of the historical instances, the issue of user interfaces becomes more significant. Yet the use of examples as a method driven by data (and not programs) does permit the system to refine its structures as the number of examples increase.

6.0 USER-COMPUTER INTERFACE TECHNOLOGIES

The INSITE system will provide many powerful functions to the user to assist them in developing time estimates for their projects. However, these functions will go to waste if the user finds it difficult to communicate with INSITE, specify the characteristics of their projects, or decipher the results. Therefore, the nature of the user interface is a critical determinant to the successful use of INSITE. This section presents the results of research conducted into several critical areas for the user interface (namely, the user-computer dialogue and user profiling).

Unfortunately, few absolute rules and guidelines are available to assist the system developer in designing interactive interfaces. This section will present "guiding principles," but it is expected that the exact nature of the user interface will evolve in conjunction with the development of the INSITE prototype through extensive involvement of the proposed user community and their feedback.

The discussion of user interfaces assumes that INSITE will be developed on a PC class of microcomputers and that the microcomputer is equipped with 1) a color monitor capable of bit-mapped graphics and multiple windows and 2) a pointing device such as a mouse. Bit-mapped graphics permit every screen dot to be individually turned on or off by resetting the corresponding bit in memory and permit the rapid display of graphic information. Windows provide the capability to juxtapose several screens of information (text or graphics) simultaneously and to move information from one window to another. Pop-up windows with menu selections permit the user to scan the menu options and select the desired one without having to recreate the original screen of information. Windows are also very useful for displaying help messages. A mouse is a small box (about the size of a deck of playing cards) with one to three buttons on its top face that functions as a pointing device. The mouse motion translates into corresponding movements of a pointer (e.g., a cursor) on a display. It allows the user to manipulate information and select commands or locations on the screen without having to enter specialized interface commands via the keyboard.

6.1 INTELLIGENT USER DIALOGUE

One of the key issues in designing a user interface is the selection of the techniques used for communications between the user and the computer. This not only involves selection of a dialogue type, but the development of techniques to assist and guide the user's interactions with the computer system. As described in Section 3, INSITE users will possess varying levels of computer usage skills and project management expertise. The user-computer interface (UCI) must supply a meaningful structure within which the two-way dialogue between the user and INSITE occurs.

6.1.1 Dialogue Types

The user communicates with the system by specifying commands and objects to be manipulated. There are five primary dialogue types:

1. Command-Driven,
2. Menu-Driven,
3. Question and Answer,
4. Form Fill-In, and
5. Graphics-Driven.

Each dialogue structure type has a particular user appeal, depending on a user's knowledge of the system and computer expertise. Many systems are now based on a hybrid of these techniques.

6.1.1.1 Command-Driven Dialogues. Command-driven dialogues typically display a brief prompt to the user (e.g., a question mark) and expect the user to type in a command name, phrase, or associated mnemonic (e.g., PRINT or PR). Since the system offers very few prompts or choices, the user is expected to know which system features are currently available and their syntax. If the system does not recognize a command that the user enters, it typically responds with an error message. The biggest advantage of a command-driven interface is speed. Very few keystrokes are required to initiate any action, and it eliminates stepping through multiple levels of menus to specify an action. Command-driven interfaces appeal to experienced users since the system functions are more readily accessible. Command dialogue's biggest shortcoming is its inherent "unfriendliness." A system or computer novice viewing a display with nothing but a prompt has no guide as to what to do next. Novice users have many problems learning a command dialogue system since they are unfamiliar with both the functions that can be accomplished and the command names required to invoke the functions. Additionally, in order to invoke a command, the user has to first remember the designated command. If there are too many commands to be able to remember them easily, users tend to find this facility too frustrating and time-consuming to use.

6.1.1.2 Menu-Driven Interfaces. Menu-driven interfaces display every possible choice that the user can make in a menu that is typically arranged in an hierarchical manner (i.e., one choice or action must be taken before another). The selection of one menu item generates another menu, which brings up yet another until a final selection allows the desired function to be accomplished. The major advantage provided by menu dialogues is the ability to guide a user through the steps needed to accomplish a task. Menus reduce memory demands because they only require the user to recognize rather than recall the correct option (Martin, 1973). However, menu-driven interfaces are not without problems. New users might find that learning larger systems is difficult because information must be integrated across a series of displays. As each menu is viewed in isolation, relationships between menus are difficult to grasp and the user may get lost in the hierarchical structure. However, menu interfaces do facilitate use by novices. On the other hand, experienced users are often frustrated when they must step through a number of menus before reaching the desired function since not all functions are available from all menus.

A strategy that is effective for workstations with display windowing and mouse capabilities is the pull-down menu that combines command- and menu-driven techniques. Pull-down menus display a command bar at the top of a window with a submenu displayed in a new window box beneath it showing all available options. The user can use the mouse to move a pointer to one of the choices displayed in the box. As the pointer moves among the menu options, the current selection is highlighted and the user depresses a keyboard or mouse key to indicate the desired selection.

6.1.1.3 Question and Answer Dialogues. Question and Answer (Q&A) dialogues present the user with a series of questions to which the user responds one at a time. The Q&A process is repeated until the system has received the necessary information. Q&A dialogues typically decide the next question based upon the answer(s) to the previous question(s). Some incorporate a degree of natural language capabilities in order to avoid simple yes/no responses. If the system cannot understand a response or requires additional information, clarification questions may be generated. Similarly, if the user cannot understand a question, an explanation facility is typically provided.

Q&A dialogues are most successful with novices who are unfamiliar with the problem to be solved. However, experienced users quickly become impatient when forced to step through a lot of questions. One way to alleviate this problem is to provide multiple modes of use (e.g., full sentence mode and abbreviation mode). Additionally, default response can be set for the particular user (this is described in more detail in Section 6.2) as part of their "user profile." An additional consideration is how to permit the user to change the response to a previous question.

6.1.1.4 Form Fill-In Interfaces. Form fill-in (or fill-in-the-blanks) interfaces provide the user with input forms in which the user enters necessary commands and data. A display of labeled fields and an area for entry of input are shown, and the user moves the cursor between the input areas and enters the appropriate information. This type of design is well-suited when there is a correspondence between the input display and paper forms familiar to the user. This type of interface requires the user to be cognizant of the field labels, permissible field values, and data entry techniques to be employed in moving among the displayed fields. Form fill-ins are most appropriate for frequent users.

6.1.1.5 Graphics-Driven Interfaces. Graphics-driven interfaces are a fairly recent development and generally provide an interface that is both "friendly" and non-restrictive. Graphic icons replace or supplement words as command designators in menu-based systems. Instead of a temporal order display such as a menu, the user is presented with a spatial order of possible actions that are represented by "iconic" or pictorial representations of actions. The user positions a graphics pointer (such as a mouse) over the icon representing the desired action. Icons take advantage of the human ability to discern pictorial differences more quickly and easily than textual differences. However, the icons must be designed carefully in order to maximize their usefulness and are best suited when a limited set of clearly distinguishable options are available. The visual interfaces made possible by iconic representation provide an object orientation rather than a procedure orientation and enhance the user's knowledge about the system and its capabilities. However, there currently exists a very limited body of information about the effectiveness of iconic representations and how they should be designed to maximize information transfer.

6.1.1.6 User Dialogue Recommendations. It is recommended that the initial dialogue style for the prototype INSITE system be a combination of question and answer and menus as illustrated in Section 7. Menu bars can be used to show a list of available INSITE functions on a line at the top of the screen. Pull-down menus can list submenus in a separate window displayed beneath the menu bar and will contain the list of commands available for a particular selection from the menu bar. Once the user has specified the INSITE function, additional dialogue with the user will occur using a series of questions. Since the series of questions will vary, depending upon the responses to previous questions, the form fill-in type of input does not appear to be appropriate. However, recent research in the area of input templates such as those used on the Apple Macintosh computer will be considered in the development of the user interface (Smith et al., 1982; Norman and Draper, 1986).

6.2 USER PROFILES

Typically, each user of a computer system develops a personal methodology for interconnecting seemingly isolated techniques and strategies in using a specific system. Over time, the user develops a great deal of problem-specific declarative and procedural knowledge; and as a result, the user becomes proficient in operating that particular system. During an individual's tenure in a specific position, he or she is exposed to various computer systems. The time and effort spent to learn a new system can be significant. The learning phase is not productive and can be considered "lost" time. If the "time to learn" could be reduced to a reasonably short amount of time, the "lost" time would be negligible and the user would be able to benefit more quickly from the system. Each user will also be applying the system to the needs of a particular project. A person oriented towards software projects will probably have a different focus than a person oriented towards hardware. Since INSITE will be applicable to all project types, it is important for it to include a capability to communicate with the user using a unique terminology applicable to the user's needs. For these reasons, it is recommended that INSITE include the capability to construct a user profile. The user profile module will be a tool designed to capture user preferences, store them, represent them, and permit the user to interact with INSITE in a customized manner that accommodates that particular user's knowledge rather than being restricted by system-defined commands and protocols.

This user profile capability will permit a computer user to tailor the INSITE interface by allowing the user to specify the terms used for communicating with INSITE using his/her particular jargon. This tailoring tool would also significantly reduce the training time required and therefore reduce the cost associated with training. For novice users, user profiles can assist in eliminating computer anxiety so that the individual will be willing to learn a new system rather than expend time and effort avoiding it. For the expert user, it will allow them to use their personal preferences in constructing their user protocol which will result in increased productivity.

Once the issues that need to be addressed to initially construct the user profile have been resolved, the potential techniques employed to cause the system to recognize a particular user and then load the user's profile into the system's working memory will need to be explored. Issues concerning maintaining, updating, and reconfiguring a user profile, once it has been initialized, will require investigation with regard to the various functions of the operating systems — the issues of compatibility, usability, and portability.

There are a few techniques currently available that allow a user to define system commands through the use of personal preference (such as macro and command files). However, such files are awkward to construct and may not be obvious to the infrequent or inexperienced user. Since many users do not possess a programming background, a criterion level of understanding needs to be established and then the degree of user control can be investigated. In addition, other variables (such as hardware limitations) may affect the degree of user freedom possible, and therefore, factors such as these need to be examined.

The exact nature of the user interface will also be considered in conjunction with system requirements. Sophisticated user interfaces frequently make heavy demands on the system, and trade-offs may be required to ensure that the benefits of the user interface are not nullified by slow system response time. The design of the user-computer interface will proceed iteratively in conjunction with the development of the INSITE prototype and will involve early and frequent interactions with the prospective user population.

7.0 INSITE SYSTEM

The INSITE system will provide support to managers performing planning and scheduling tasks which require estimations of project/task duration. The system will be configured as an estimation workbench and will reside on a microcomputer. In the sections which follow, the high-level system design of INSITE will be presented and the major system modules described.

7.1 HIGH-LEVEL DESIGN

The four major functional components of INSITE are presented in Figure 7-1. System responsibilities are divided between these software modules as follows:

- **ESTIMATOR** — generates project/task duration estimates.
- **MEDIATOR** — manages all human/computer interface processes, including key and mouse inputs, windowing, graphics, and formatting of data and dialogue outputs.
- **HISTORIAN** — manages historical data base as well as interface and estimation knowledge bases.
- **DIRECTOR** — controls activity of all other system components and drives human/computer dialogues.

Each software module will be configured independently for maximum system flexibility. For example, moving from one computer display system to another will only require a code modification to MEDIATOR. Likewise, accessing a new data base will only require a modification to HISTORIAN. This approach will enhance both portability and the capacity for future upgrades.

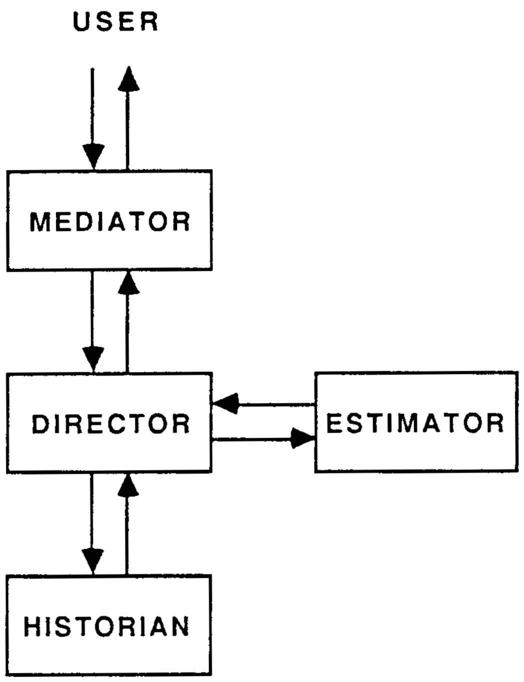


Figure 7-1. Block Diagram of INSITE System Organization

7.2 ESTIMATOR

With the exception of the ESTIMATOR module, the components of INSITE are clearly feasible in the microcomputer environment due to the availability of text and graphics output. Window and mouse interfaces are supported on almost all computers. Data base systems and expert system shells are also becoming commonplace. However, it is not immediately obvious that it is feasible to develop a microcomputer-based system which can rapidly respond (i.e., minutes) to a request for estimates. In the two subsections which follow, the feasibility of a microcomputer implementation of INSITE will be explored — the structure of the INSITE estimation process will be specified, and the preliminary development and testing of a microcomputer-based estimation algorithm will be presented.

7.2.1 Multiple-Estimate Architecture

For any given project, a time estimate could be based on a single previous project or several projects. Ordinarily, a set of relevant historical cases would be extracted from the data base. On the basis of this set, a measure of central tendency is calculated to predict project duration, providing an estimate not tied to the peculiarities of a single case. Basing an estimate on a single example would be required if the historical data base provided no other relevant cases (as might occur if the data base had only recently been established). When a single previous project becomes the only source for an estimation, dependency knowledge (stored in a project knowledge base) must be brought to bear. For example, if the source project for the estimate was a larger project than the one being estimated, the known time for the source project must be adjusted downward to produce an estimate, based on the system's knowledge regarding the impact of project size on project duration. Figure 7-2 presents a procedural schema for the INSITE ESTIMATOR module. As shown in the diagram, INSITE provides two separate components to calculate estimates. With sets of cases, a straightforward computation of central tendency is possible. With single cases, dependency knowledge must be applied. It is expected that the dependency algorithms/heuristics will be fairly limited in scope, possibly encoded as a relatively small rule base since the dependencies involved are poorly understood and do not exist at the level of detail required to perform direct calculations.

Projects in the historical data base have similar attributes to the new project being considered. The attributes recorded in the data base are selected on the basis of the underlying dependency relationship. That is, if a project attribute is believed to impact project time duration, it is included in the data base structure. Some historical cases will be very poor analogies to the new project; others will be good analogies to the extent that their attribute values vary from those of the new project. In the formulation "A is to B as C is to D", "B" and "D" represent project duration. "A" and "B" can be associated with the projects themselves or with the collection of attributes by which they are represented in the data base. The relation between "A" and "B" (and between "C" and "D") is, in fact, the complex dependency relation between project

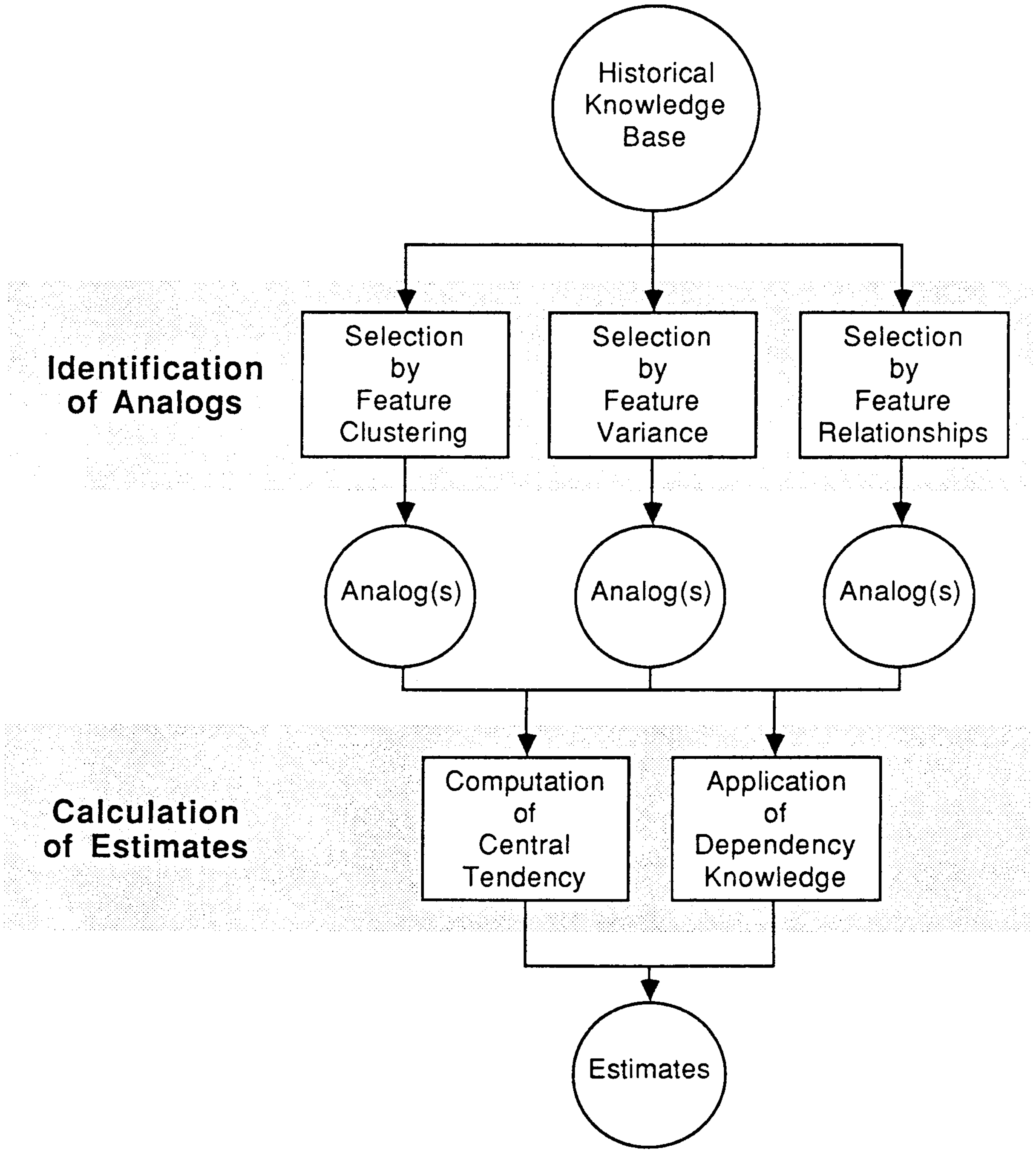

Historical Knowledge Base
Identification of Analogs
Selection by Feature Clustering
Selection by Feature Variance
Selection by Feature Relationships
Analog(s)
Analog(s)
Analog(s)
Calculation of Estimates
Computation of Central Tendency
Application of Dependency Knowledge
Estimates

Figure 7-2. Schema of ESTIMATOR Module

attributes and duration, represented by the attribute values. The identification of good analogs in the data base is a matching problem, and this is the central problem addressed by the ESTIMATOR module. As shown in Figure 7-2, three separate analogy identification algorithms will be incorporated into INSITE.

The **feature clustering method** of selecting analogs for use by the estimate calculator will use pattern matching algorithms to define project clusters based on project attributes. The project to be estimated will then be matched and placed into one of these clusters. The analogs used will be all of the projects within that cluster. As a method of finding only one analog, the projects will be seen as a point in an n-dimensional space (where n is the number of features defined for a project); the analog project will then be the nearest point to the project to be estimated.

The **feature variance method** of analog identification defines a class of analogs using a threshold variance from the project to be estimated for each of the attributes defined for the project data base. This differs from selection by feature clustering. In this case, the groupings are created by the thresholds defined (that is, a "natural grouping" is not needed). This method is described in detail in Subsection 7.2.2 below.

The **feature relationships method** defines a set of analogs via closeness of feature *relationships* (as opposed to closeness of *features*). For example, given the following subset of a data base:

PROJECT	PROJECT SIZE	STAFF SIZE
1	100	10
2	2000	200
3	8000	800

Selection by feature clustering or selection by feature variance may not find an analogy between projects 1, 2, and 3. However, the analogy is obvious to the human observer. Selection by feature relationships seeks to capture that feeling of "sameness" that is easy to see, but may be hard to describe. In this case, all three projects would have exactly the same "project size to staff size" relation (i.e., "10"). In selecting a group, some threshold difference would be defined (as with selection by feature variance). When selecting only one analog, the "closest" match would be selected.

Since there is no way of determining which identification algorithm is best, and since it is unlikely that a single algorithm will be superior in all cases, all three algorithms will be implemented within INSITE. ESTIMATOR will therefore potentially produce six separate estimates, based on the combination of identification algorithms and calculation methods — set-by-clustering, single-by-clustering, set-by-variance, single-by-variance, set-by-relationship, and single-by-relationship (six separate paths through the schema in Figure 7-2). Various methods can be used to control or limit the presentation of this information, including confidence levels on the estimates, size of the analog set (if very small, the single-case strategy should produce better results), and user intervention. It is also possible to perform calibration runs to assess relative estimation accuracy for the six separate estimation strategies in combination with a specific data base. That is, each case in the historical data base can be used in turn as the "new" project and an estimate produced, based on the other cases. A measure of accuracy can then be calculated for each of the six algorithm combinations, according to its ability to predict times for cases already known.

7.2.2 Algorithm Design Approach

The use of analogies in INSITE is direct and computationally simple. Unlike the Generalized Analogical Reasoning System (GARS) proposed by Silverman (1985) for systems management applications, no attempt is made in INSITE to capture the entire range of human analogizing abilities. Also unlike GARS, which appears to have made little headway due to its complexity, INSITE can be readily implemented in a microcomputer environment using known technologies. In order to demonstrate the feasibility of INSITE, this section will discuss in detail the design steps, implementation, and results of a set-by-variance identification algorithm in combination with a computation of central tendency. (The development of the other two identification algorithms and the heuristics for producing an estimate from a single historical example will proceed in a similar manner during the Phase II effort.)

A program, SETBYVAR, was developed to compute time estimation based on the set-by-variance method. The program was coded and debugged using Microsoft C on an IBM AT-compatible microcomputer. First, four data bases of projects were created based on a set of rules for attribute interaction. While identical rules were used, the attribute values were generated randomly utilizing either a normal or a uniform distribution. A 100 and a 1000 item data base were created for each distribution type. This allowed an analysis of trends from the resulting data as illustrated in Figure 7-3. A sample SETBYVAR dialogue is shown in Figure 7-4. Initial results were promising, based upon a limited set of attributes and simple relationships between attributes.

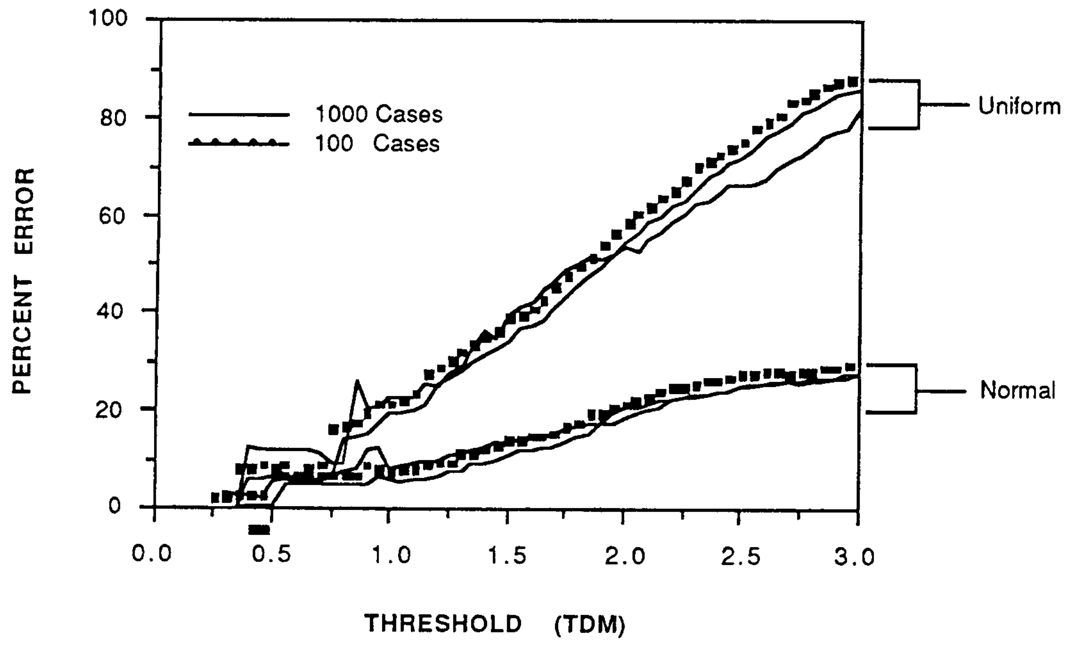
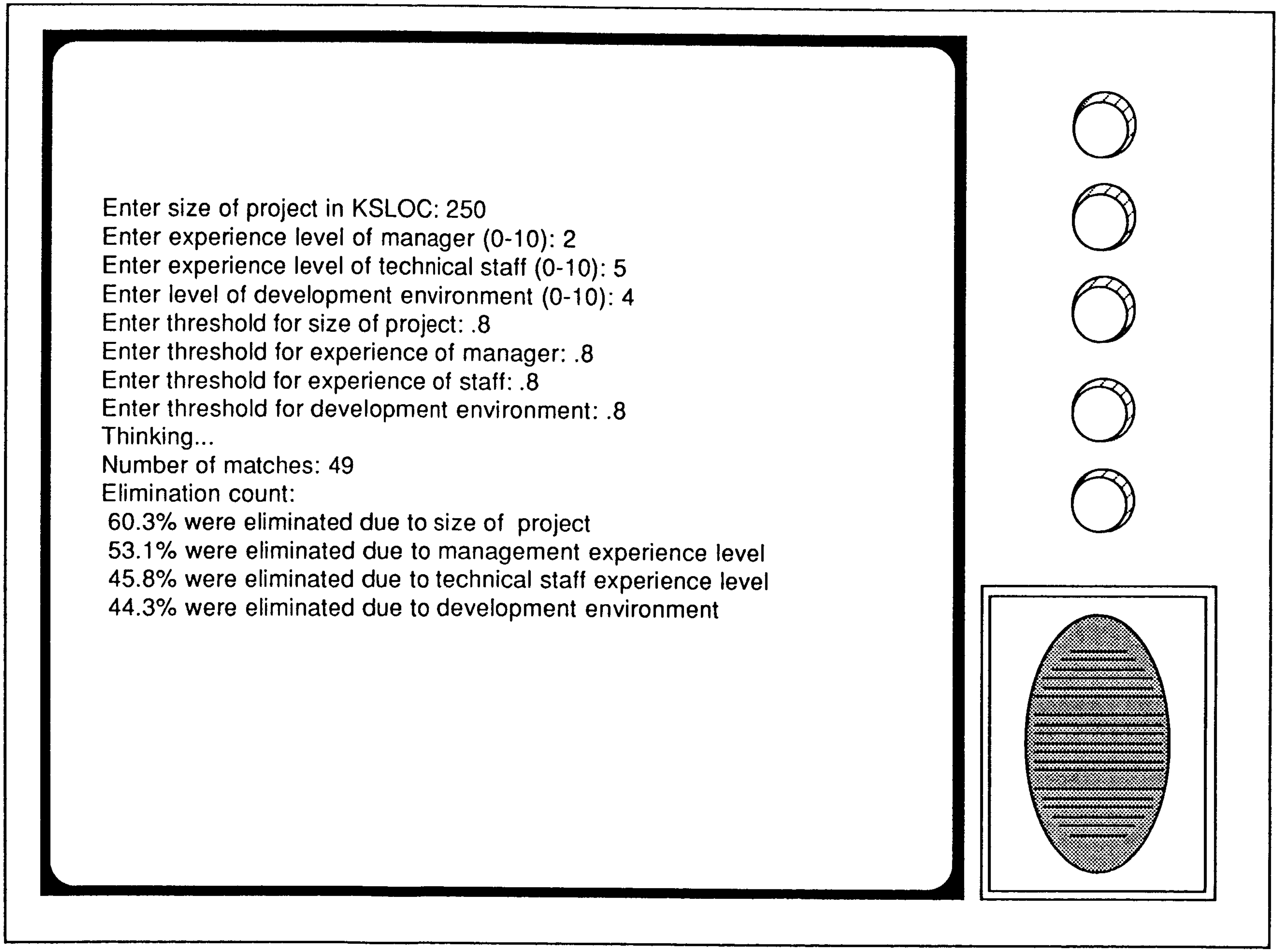
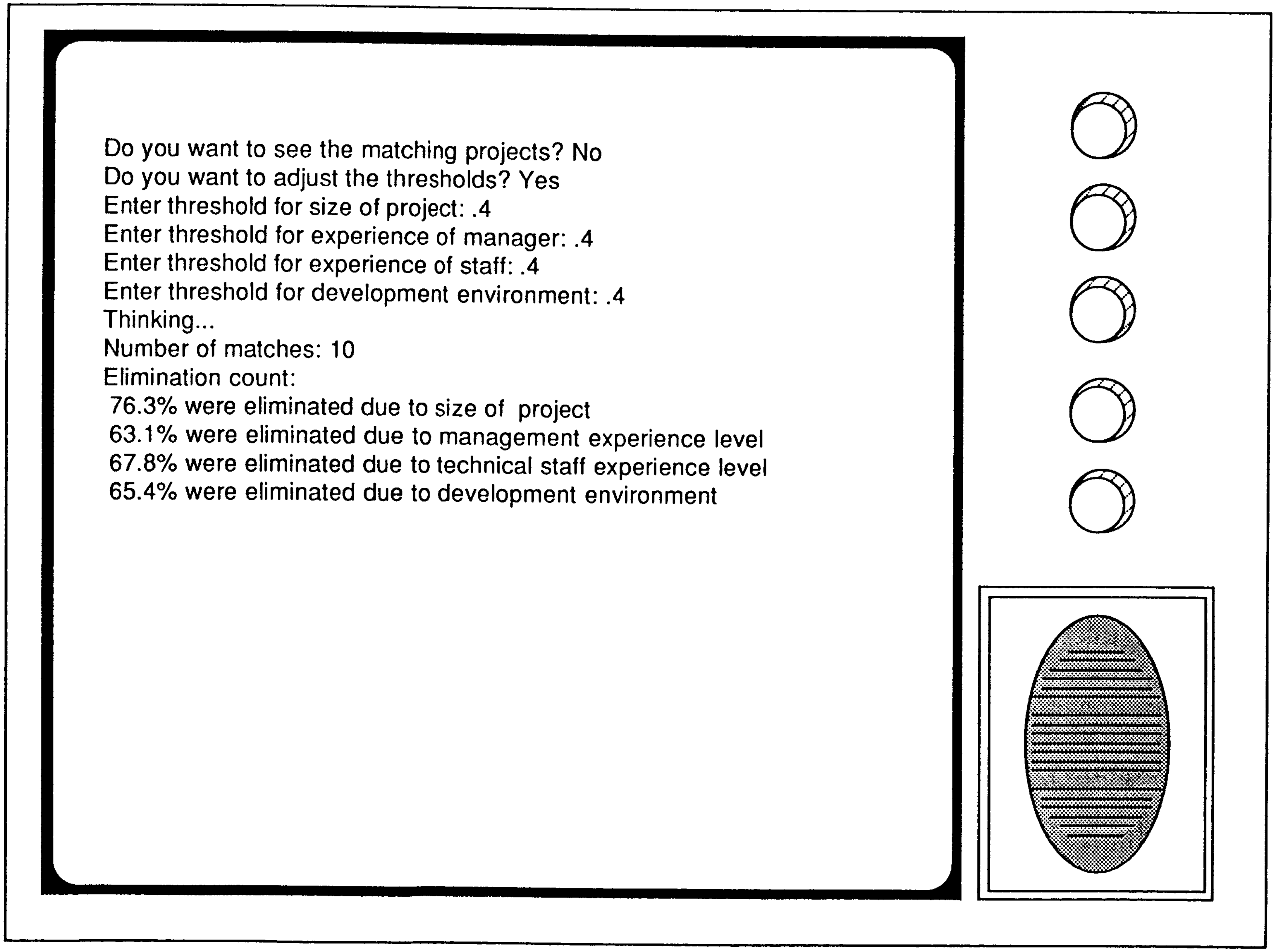


Figure 7-3. Set-by-Variance Test Runs



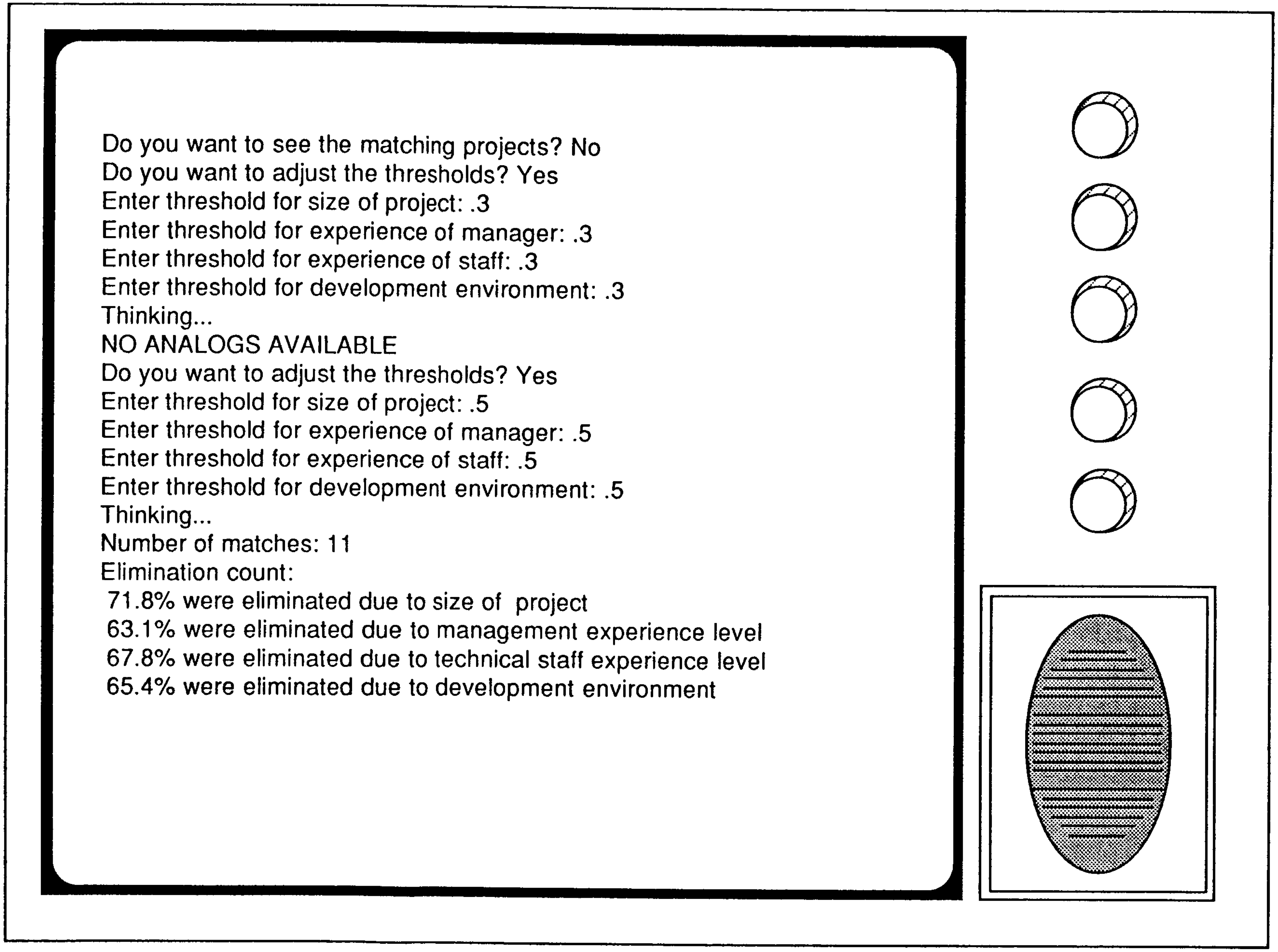
```
Enter size of project in KSLOC: 250
Enter experience level of manager (0-10): 2
Enter experience level of technical staff (0-10): 5
Enter level of development environment (0-10): 4
Enter threshold for size of project: .8
Enter threshold for experience of manager: .8
Enter threshold for experience of staff: .8
Enter threshold for development environment: .8
Thinking...
Number of matches: 49
Elimination count:
 60.3% were eliminated due to size of  project
 53.1% were eliminated due to management experience level
 45.8% were eliminated due to technical staff experience level
 44.3% were eliminated due to development environment
```

Figure 7-4. SETBYVAR Sample Dialogue



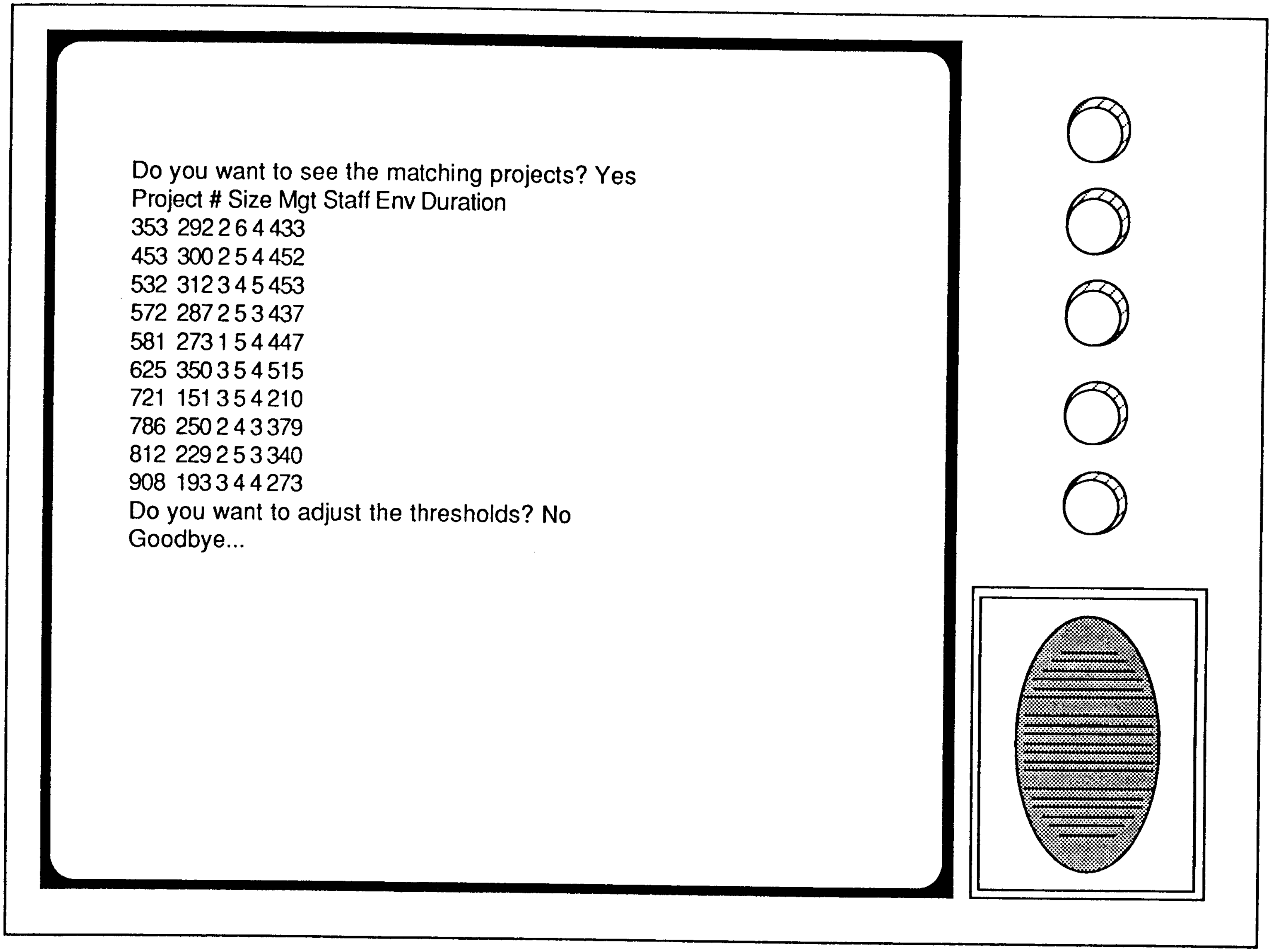
```
Do you want to see the matching projects? No
Do you want to adjust the thresholds? Yes
Enter threshold for size of project: .4
Enter threshold for experience of manager: .4
Enter threshold for experience of staff: .4
Enter threshold for development environment: .4
Thinking...
Number of matches: 10
Elimination count:
 76.3% were eliminated due to size of  project
 63.1% were eliminated due to management experience level
 67.8% were eliminated due to technical staff experience level
 65.4% were eliminated due to development environment
```

Figure 7-4. SETBYVAR Sample Dialogue (continued)



```
Do you want to see the matching projects? No
Do you want to adjust the thresholds? Yes
Enter threshold for size of project: .3
Enter threshold for experience of manager: .3
Enter threshold for experience of staff: .3
Enter threshold for development environment: .3
Thinking...
NO ANALOGS AVAILABLE
Do you want to adjust the thresholds? Yes
Enter threshold for size of project: .5
Enter threshold for experience of manager: .5
Enter threshold for experience of staff: .5
Enter threshold for development environment: .5
Thinking...
Number of matches: 11
Elimination count:
 71.8% were eliminated due to size of  project
 63.1% were eliminated due to management experience level
 67.8% were eliminated due to technical staff experience level
 65.4% were eliminated due to development environment
```

Figure 7-4. SETBYVAR Sample Dialogue (continued)



```
Do you want to see the matching projects? Yes
Project # Size Mgt Staff Env Duration
353  292 2 6 4 433
453  300 2 5 4 452
532  312 3 4 5 453
572  287 2 5 3 437
581  273 1 5 4 447
625  350 3 5 4 515
721  151 3 5 4 210
786  250 2 4 3 379
812  229 2 5 3 340
908  193 3 4 4 273
Do you want to adjust the thresholds? No
Goodbye...
```

Figure 7-4. SETBYVAR Sample Dialogue (continued)

The problem to be answered is briefly stated as follows: given an historical data base of projects and their respective durations, derive an accurate estimate of the duration of a new project (project A). Development of the algorithm proceeded as a series of questions answered as described in the following paragraphs.

<u>What is the main concept behind this algorithm?</u> The main concept is to find "similar" projects in the data base using a variance technique, then extrapolate a project duration from them for project A.

<u>What is the definition of "similar"?</u> "Similar" here means that the appropriate attributes of the project are close enough (within some threshold) to project A that they can be seen as good analogs for use in the estimate.

<u>What are the appropriate attributes?</u> The identification of appropriate attributes for the comparison of projects is one of the main tasks of Phase II of the INSITE project. These attributes must be defined before the project data base is created or populated. For this discussion, they can be assumed to be the attributes of the projects defined in the data base which has already been created. For this example, five attributes were selected: type of project (hardware, software, etc.); size of project (in appropriate units); experience of manager (on a scale of 0-10); experience of technical staff (on a scale of 0-10); and development environment (on a scale of 0-10). The type of project for this example is the development of a software testbed; size is defined in thousands of source lines of code (KSLOC); the technical staff is the programming staff; development environment is a measure of the levels of programming tools available.

<u>What are the thresholds which delimit a valid analog?</u> The thresholds are defined as a threshold deviation multiplier (TDM) times the standard deviation of the entire project data base for the attribute in question. The TDM is user-definable; for example, if the user thinks that programming staff is a much

C-2

more important factor than management experience, he might assign a TDM of 0.8 to the former and a TDM of 1.2 to the latter. This will be discussed further below.

How are the analogs culled from the data base? The data base is searched, looking for a set Z of projects, as follows:

$$
Z = \{ x \mid \begin{array}{l} |P1_x - P1_a| <= TDM1(S1) \text{ and} \\ |P2_x - P2_a| <= TDM2(S2) \text{ and} \\ \quad\bullet \\ \quad\bullet \\ \quad\bullet \\ |Pn_x - Pn_a| <= TDMn(Sn) \} \end{array}
$$

where:

n = number of attributes
a = the project to be estimated
Sx = standard deviation for attribute x
Pn_x = the value of attribute n for project x
TDMx = threshold deviation multiplier for attribute x

which defines the set of projects to be used as analogies for the estimate.

How is the time estimate calculated? After the set of valid analogs is defined, the estimated duration (value) of the project is calculated as follows (ε indicates set membership):

$$
V_{a\ est} = \frac{\sum_{b=1}^{n(Z)} \quad V_b \ \varepsilon \ Z}{n(Z)}
$$

where:

a = project to be estimated
V_X = duration of project x
n(Z) = number of elements in the set Z

The estimate also includes a range of error value as in:

ESTIMATE: 814 months plus or minus 66 months

which is defined as:

ERROR = (average TDM x std dev)

where average TDM is the average of the threshold deviation multipliers entered by the user, and standard deviation (std dev) is the standard deviation of the time attributes for the historical data base.

What is the optimum value for the TDM's? This can only be answered in a very general sense since the TDM's are user-controlled. However, some guidelines are available:

- Multipliers less than 1.0 often show erratic behavior (since the list of analogs becomes small), and estimates can therefore vary widely. Taken to an extreme, the estimate is calculated from a sample of one, which makes it very volatile. Pushed too far, small multipliers cause a "NO ANALOGS AVAILABLE" response from INSITE, thus giving no answer at all.

- Multipliers larger than 1.0 begin to cause a linear decay in the accuracy of the estimate. Taken to an extreme, large multipliers cause the estimate to be made based on the entire data base, thus returning as an answer the average duration of all projects in the data base.

These results are shown graphically in Figure 7-3. It shows that as threshold increases above 1.2, the error increases. It also shows erratic behavior below 1.0 as well as no answers at all (therefore, there was no error) below 0.5. This suggests that an initial value between 0.8 and 1.2 for the multiplier is useful, with further tuning made by the user afterward.

What is the effect of the size of the data base on the estimate? Intuitively, one would expect that the estimates would become more accurate as the size of the example data base increases, and this is the case for the samples that we used. The real effect of a larger data base is, however, that the error curve is smoothed. These results are also shown graphically in Figure 7-3.

What is the effect of the distribution of the data on the estimate? As shown in Figure 7-3, normally distributed data (the bottom two curves) results in much more accurate estimates than does uniformly distributed data (the top two curves). This is not difficult to explain since the calculation is based on standard deviation, which is defined only on normally distributed data. It is also not a particularly restrictive requirement since one would expect that all projects at a particular site would be "balanced" around some central tendency figure. The fact that the calculation yields meaningful results on uniformly distributed data as well is important, since such a case can certainly be imagined.

What assumptions are made about the data? The only assumption made about the data is that there is *some* relationship between the attributes and the time value. The usefulness of this approach over others (for example, the parametric approach) is in the fact that this relationship *need not be known.* The data used in our example of a software project has a fairly complex, though not unreasonable, relationship. This relationship is defined by the following rules:

- The time to perform a project is a function of its size in KSLOC.
- A poor manager adds time to a project, but this effect is reduced if the technical staff is good or excellent.

- A poor development environment adds time to a project, but this effect is similarly reduced by the presence of a superior technical staff (i.e., they would build their own tools).

How reliable are the estimates? At thresholds between 0.8 and 1.2 standard deviations, the errors in estimates were all less than 20% (and often below 10%) with linear correlation coefficients between 0.94 and 0.998. With all four types of data samples, across thresholds ranging from 0.5 to 3.0 standard deviations, the actual value for the project was within the predicted error range 99.95% of the time (one case in 2200 tests was outside of the range). Of course, the data used contained no noise component since the time values were wholly dependent on the attribute values. Whether results this good can be obtained using "real-world" data is dependent on the selection of appropriate comparison criteria (the attributes of the projects which are stored in the data base) and the strength of the relationships between the attribute values and the time values.

What are the limitations of this approach? As described above, there must be some relationship between the attributes of projects and their duration. For example, position of the moon on the first day of the project could probably be shown not to have an effect on the outcome of the project. Having this field in the data base would throw "noise" into the calculations, thus making the estimates less accurate. Careful selection of the criteria for the calculations is imperative. Another limitation with the method is that estimates tend to be arbitrary as the data base tends toward zero elements. However, even with the limited (15 item) data base used to test the software cost estimation models cited above (Section 5.1), average error was 53% when we used this method, not 460% as given by the other methods (however, some projects could not be estimated due to lack of analogs). This method is also limited in that it will sometimes return no estimate at all, due to lack of analogs. This is either a limitation or a feature: should a system return an answer — *any answer* — no matter how bad? Or should it be intelligent enough to "know that it does not know"? The discussion of an optimal system suggested that the latter choice was preferable.

7.3 MEDIATOR

Since the INSITE interface is dependent on both user characteristics and project attributes, which will only be specified during the course of knowledge engineering in a Phase II effort, it is premature to present an interface design at this stage of development. However, since the intent is to rely on rapid prototyping, it is possible to present a plausible first-cut configuration which could be constructed quickly, given a reasonable development environment. In the paragraphs which follow, one simple interaction sequence is presented which integrates many of the interface technology options discussed in Section 7. It does not represent a final design, but rather a potential starting point. (As shown in the accompanying figures, a Macintosh-like environment is assumed.)

When the user first enters INSITE, a menu bar appears at the top of the screen (see Figure 7-5). This menu bar remains in place throughout a session and gives access to INSITE functions as follows:

- **File** — permits storage, retrieval, and examination of estimation runs.
- **Proj Spec** — allows entry, modification, and examination of project attributes for the project to be estimated.
- **Estimate** — gives access to estimation parameters and results and allows the user to initiate estimation runs.
- **Data Update** — allows direct access to the historical data base for entry, modification, and examination of project data.
- **Help** — permits entry into help utilities for all system activities.
- **Quit** — exits INSITE.

Menu bar selections are made by moving the mouse cursor over the label of interest and holding down the mouse button. A pull-down menu then appears, from which the required function can be selected. In a typical interactive session, the user might first wish to enter the attributes of a project to be estimated. This is accomplished by placing the arrow-shaped mouse cursor over the "Proj Spec" label in the menu bar, depressing the mouse button ("clicking"), and selecting "Edit Attributes" from the menu which appears. A Project Specifications Window will immediately appear, with the text "What type of project?" in its upper left corner (see Figure 7-5). The user responds by entering the type name. This is the first exchange in a dialogue which establishes the attributes required by the estimation process. The sample of dialogue presented in Figure 7-5 includes examples of:

1. **Smart data entry** — the system recognizes simple type-in errors like "communications" and suggests a correction.

2. **User profiling** — the system establishes a meaning for the user term "large" by referring to previous projects the user is familiar with.

3. **On-line help** — the system is able to elaborate its question if the user requires more information.

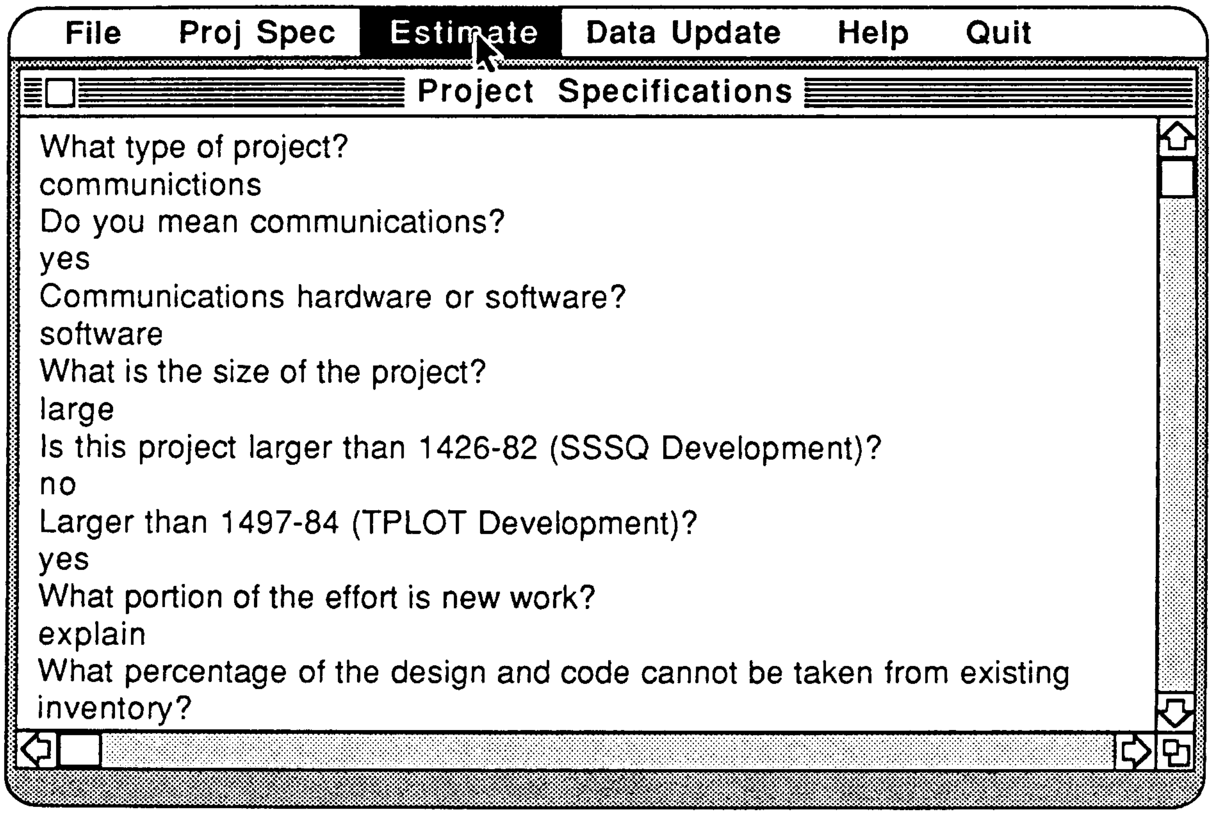
The Project Specifications Window is scrollable and allows the user to examine the entire dialogue session at any time. In Figure 7-5, the user has scrolled back to the top of the session after completing the specifications. The mouse cursor has been moved over the "Estimate" label in the menu bar in preparation to initiate an estimation run.

After running the ESTIMATOR software and seeing the estimate results, the user may be interested in examining the set of analogs selected by a particular algorithm. In this case, the user clicks on the "Estimate" label in the menu bar and selects "Variance Set" from the pull-down menu. In response, the system displays the Analogs Window as seen in Figure 7-6. This window presents the characteristics of the set, including the thresholds for each attribute as established by the user, as well as a scrollable list of the projects in the set.

Each project in the set is identified by information which is likely to enable the user to map the list to his or her actual recollections. In the figure, the user has placed the mouse cursor over the name of a project of interest. Clicking on this field causes the system to display a Project Attributes Window (see Figure 7-7).

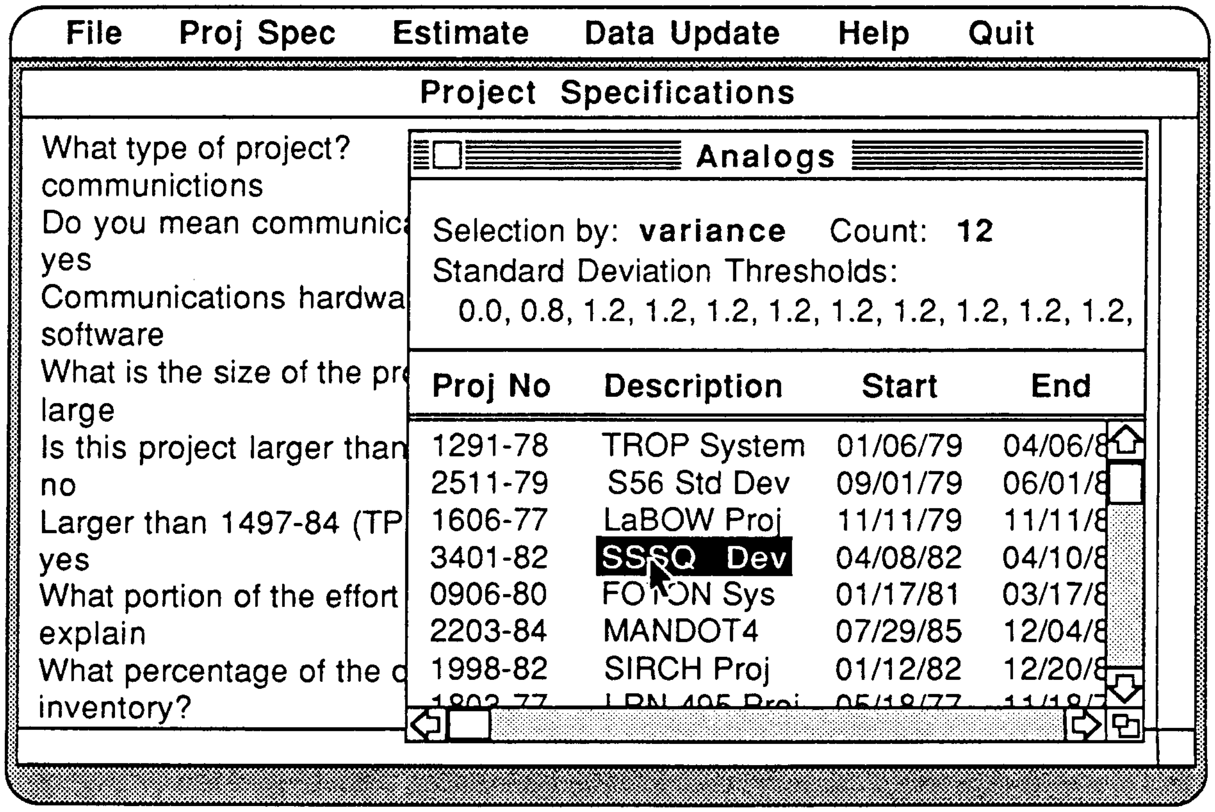
The Project Attributes Window allows the user to examine all of the attribute values for a specific historical case and compare them to the new project being estimated. Since the actual significance of attributes will not necessarily be obvious from the label assigned, embedded help messages are available. If the mouse cursor is placed over a piece of text connected to a help message, that text will switch to reverse video. In Figure 7-7, the user has positioned the cursor over the label of the project magnitude attribute. If the mouse button is then held down, a help box appears containing an explanation of the term (see Figure 7-8).

It should be apparent that INSITE is not intended to simply supply a single answer to a single question. INSITE is a workbench environment for the estimation of project duration. The user must be involved in the estimation process and ultimately make the estimation decision. The system provides tools for building an estimate based on historical records. If needed, the user may take the best estimate (as understood by the system) without question. However, INSITE will be a more useful tool if the user explores the system's alternative solutions and examines the source data for estimates of interest. The system will have more historical data available than any single user and will, in that sense, be a superior estimator. However, the system will only capture part of the human expertise involved in the estimation process and must, therefore, be provided with an interface which will allow it to function as part of a human/machine estimation team.



What type of project?
communictions
Do you mean communications?
yes
Communications hardware or software?
software
What is the size of the project?
large
Is this project larger than 1426-82 (SSSQ Development)?
no
Larger than 1497-84 (TPLOT Development)?
yes
What portion of the effort is new work?
explain
What percentage of the design and code cannot be taken from existing inventory?

Figure 7-5. Project Specifications Window



Project Specifications

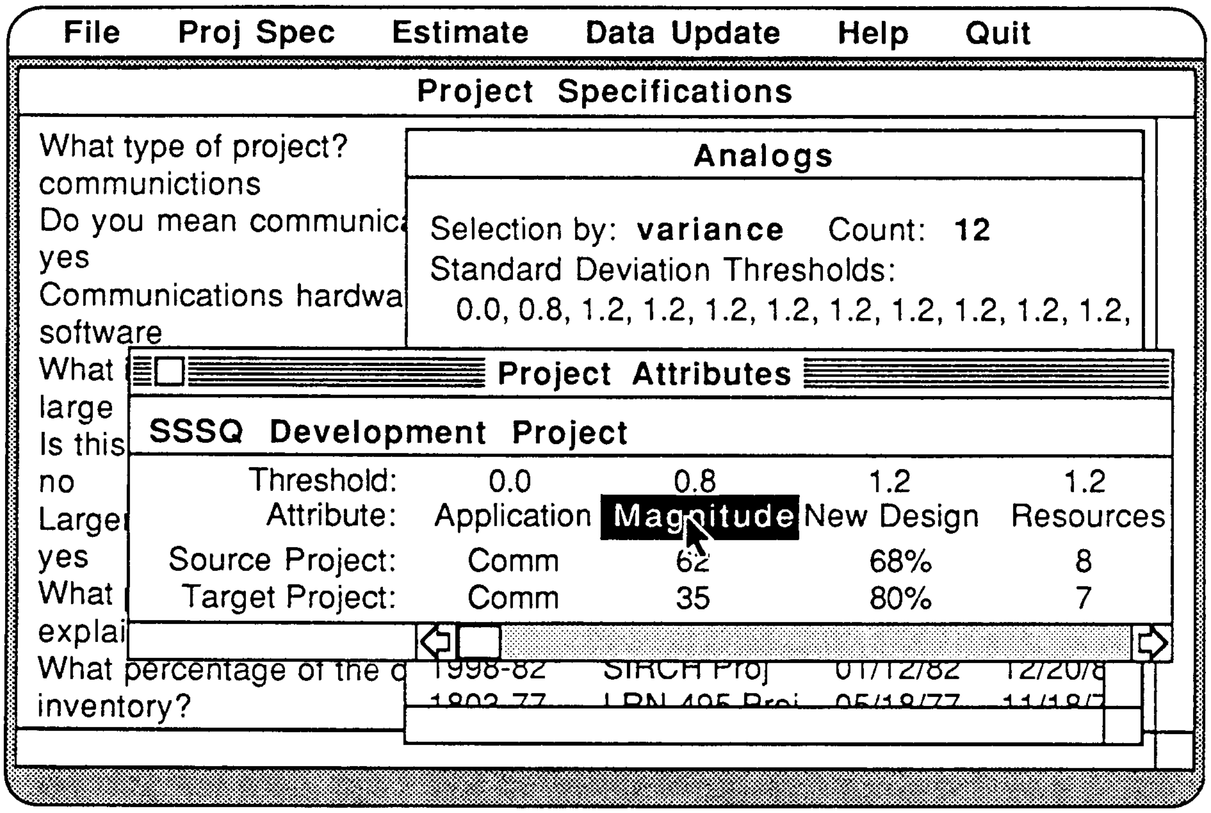
What type of project?
communictions
Do you mean communic
yes
Communications hardwa
software
What is the size of the pr
large
Is this project larger than
no
Larger than 1497-84 (TP
yes
What portion of the effort
explain
What percentage of the c
inventory?

Analogs

Selection by: **variance** Count: **12**
Standard Deviation Thresholds:
0.0, 0.8, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2,

Proj No	Description	Start	End
1291-78	TROP System	01/06/79	04/06/8
2511-79	S56 Std Dev	09/01/79	06/01/8
1606-77	LaBOW Proj	11/11/79	11/11/8
3401-82	SSSQ Dev	04/08/82	04/10/8
0906-80	FOTON Sys	01/17/81	03/17/8
2203-84	MANDOT4	07/29/85	12/04/8
1998-82	SIRCH Proj	01/12/82	12/20/8

Figure 7-6. Analogs Window



Project Specifications

What type of project?
communictions
Do you mean communic
yes
Communications hardwa
software
What
large
Is this
no
Large
yes
What
explai
What percentage of the o
inventory?

Analogs

Selection by: **variance** Count: **12**
Standard Deviation Thresholds:
0.0, 0.8, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2,

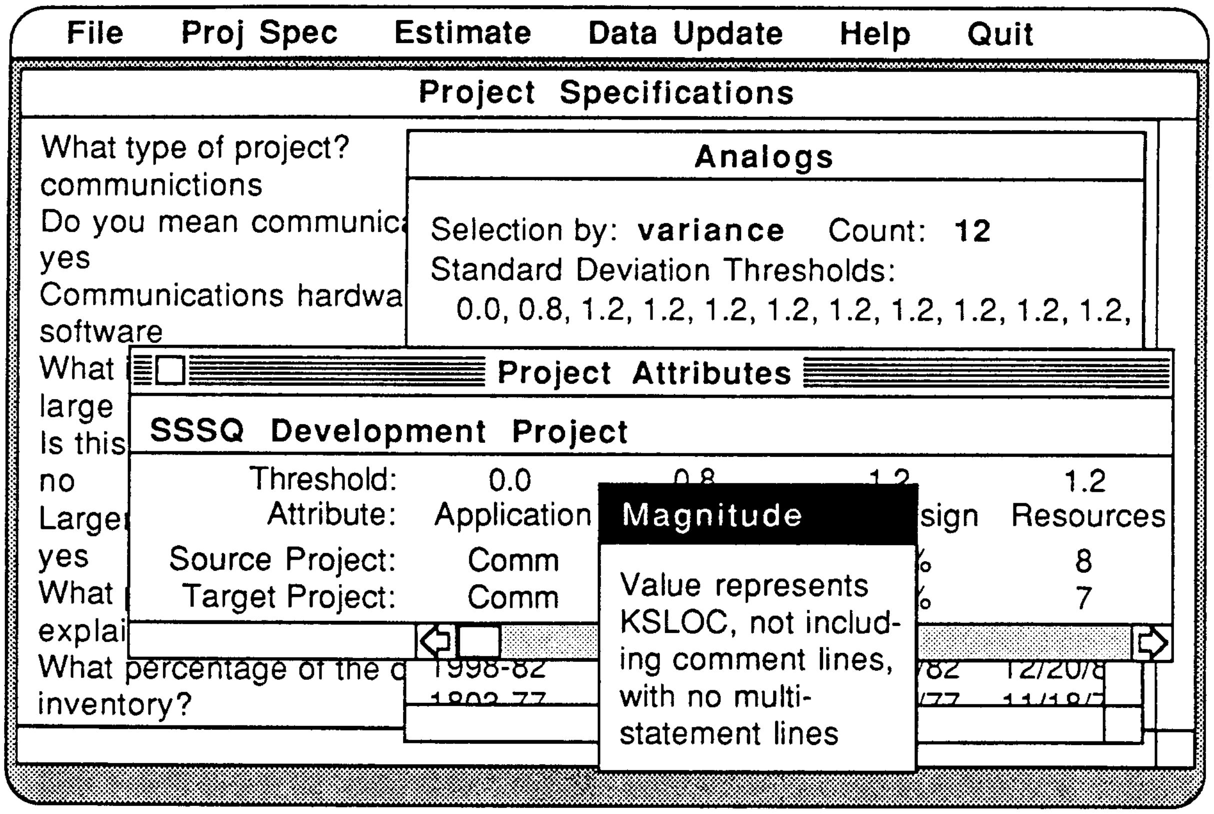
1998-82 SIRCH Proj 01/12/82 12/20/8
1802-77 LRN 495 Proj 05/18/77 11/18/7

Project Attributes

SSSQ Development Project

Threshold:	0.0	0.8	1.2	1.2
Attribute:	Application	Magnitude	New Design	Resources
Source Project:	Comm	62	68%	8
Target Project:	Comm	35	80%	7

Figure 7-7. Project Attributes Window



File Proj Spec Estimate Data Update Help Quit

Project Specifications

What type of project?
communictions
Do you mean communic
yes
Communications hardwa
software
What
large
Is this
no
Large
yes
What
explai
What percentage of the c
inventory?

Analogs

Selection by: **variance** Count: **12**
Standard Deviation Thresholds:
0.0, 0.8, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2,

Project Attributes

SSSQ Development Project

Threshold:	0.0		1.2
Attribute:	Application	sign	Resources
Source Project:	Comm	%	8
Target Project:	Comm	%	7

Magnitude

Value represents KSLOC, not including comment lines, with no multi-statement lines

Figure 7-8. Magnitude Help Box

7.4 HISTORIAN

The HISTORIAN module will control access to all system data. There are three information components necessary to the successful operation of INSITE — historical project data, project knowledge, and user knowledge. In order to process this information, INSITE will include both data base software and knowledge base software. A knowledge base is distinct from a data base in that a knowledge base incorporates not just data, but the relationships among the data, the relationships among the types of data, and further, the relationships among the relationships among the data and types of data.

The INSITE Historical Project Data Base will utilize standard data base management technology. The attributes required to characterize project history can easily be arranged in table format, with rows representing projects and columns representing their attributes. This structure can be readily accommodated using relational data base management systems. The technology driving this aspect of HISTORIAN is well-understood and readily available in "off-the-shelf" software packages. It is expected that a working historical project data base will require a relatively small number of cases (i.e., hundreds) and that, therefore, the analog identification process will not be degraded by microcomputer limitations.

Project and user interface knowledge require a totally different management technique. The INSITE Project Knowledge Base contains the kind of information required by INSITE concerning projects, their characteristics, classification scheme, component structure, and time estimation dependencies, and how that information needs to be encoded. The INSITE User Knowledge Base contains the information concerning the user which will be required for user profiling and the intelligent dialogue aspects of the system. This type of information requires a knowledge management system (such as those used in production rule knowledge schemes). For example, in the case of user profiling, the interpretation of a particular user's terminology can be expressed as in the following case:

IF project.size = **large** THEN lines_of_code > 50000

IF project.size = **average** THEN 50000 > lines_of_code > 10000

IF project.size = **small** THEN lines_of_code < 10000

This type of knowledge can be encoded using available knowledge representation technology. The knowledge scheme developed for INSITE must be capable of representing the full range of knowledge required by the system, be easy to use, and provide the flexibility to be applicable to a wide variety of applications.

Information encoded for use by HISTORIAN will derive from several sources. System knowledge of project attributes and estimation heuristics (required for estimation from single historical cases) must both be incorporated into the basic structure of INSITE. The knowledge itself must be elicited from experts in project planning/scheduling during the course of system development. The historical project data base is specific to a single installation or user community. Historical data may be entered by users as part of the installation process, making the system immediately usable for estimation, or built up over time by the entry of data as projects are completed. Profiling information is specific to a single user, but it is expected that the acquisition of this data will occur during system use and be largely transparent to the user as noted in the discussion of MEDIATOR.

7.5 DIRECTOR

As its name implies, the DIRECTOR module of INSITE will control the flow of information between the MEDIATOR, ESTIMATOR, and HISTORIAN modules. It will decide what needs to be done, when it will be done, and who (what module) has to do it. All interactions between the other three modules will flow through the DIRECTOR; this will facilitate both system implementation and system enhancement by keeping the functions of each module well-defined. Interaction will occur via calls to the DIRECTOR as in:

CASE = DIRECTOR (HISTORIAN, NEXT-ANALOG, CURRENT-CASE)

DIRECTOR will have the same basic use to INSITE that an operating system has to a computer program; it will control all of the aspects of the system while allowing the other modules to perform their own functions. It will contain the access paths to the "overhead" types of functions that each module will need, either by directly controlling the process or by calling on another module to do so. This design strategy will enable the components of INSITE to be developed separately with a minimum of integration headaches when the system is connected.

8.0 PHASE II DEVELOPMENT

8.1 PHASE I FEASIBILITY ASSESSMENT

The Phase I assessment has been completed, and the objectives stated in the original proposal have been met. The Integrated NASA System for Intelligent Time Estimation (INSITE) concept is assessed as feasible. The detailed investigation and analyses surrounding this innovative concept have produced the following conclusions:

- Accurate time estimation is critical to effective project management at NASA.
- Time estimation, as performed currently, is a complex, error-prone process.
- Current technologies can be combined to produce an automated tool to be used by project managers for time estimation at all levels of development.
- User profiling and intelligent dialogue systems will enhance the usability of the INSITE system.

The primary risk areas associated with the development of the INSITE system are in the development of the estimator algorithms, the population of historical project data bases, and the ability of the host system to support the user interface.

The level of risk associated with the selection and incorporation of the estimating models is minimal. The partial proof-of-concept experiment indicated that an algorithm can be developed that yields results which are better than those produced using many current estimation techniques. Further, additional models can be incorporated as deemed appropriate and existing models modified on a continuing basis.

Based on the above findings, the concept is judged feasible with a low level of risk. The Phase II effort will focus on the development of an INSITE version to deal with an aspect of project management at NASA as defined during the first task in conjunction with the panel of NASA experts.

8.2 GOALS OF PHASE II RESEARCH EFFORT

The primary goal of the Phase II research effort is to develop a prototype version of INSITE. The primary emphasis of the Phase II effort is on the system design and prototype implementation. The prototype development process will culminate with an actual estimation of a current NASA project.

8.3 PHASE II OBJECTIVES

The primary objective of the Phase II research effort is development and demonstration of the prototype INSITE system. The effort will involve knowledge acquisition; hardware and software installation; detailed system design; system evaluation; system development; and system documentation. Specific technical objectives for each of these areas are presented below.

Task 1: Knowledge Acquisition

- Interview experts — Analytics and NASA will identify three to five project scheduling experts, set up an expert panel, and interview each expert separately.
- Compile baseline information — The development team will analyze and combine the information culled from the expert interviews into a cohesive form.
- Review with experts — After the baseline information is compiled, Analytics will then review it with the entire panel of experts.
- Refine project knowledge — The project information will be refined based upon the comments elicited from the panel of experts.

- Coordinate project data collection — After the appropriate types of project information are identified, the team will obtain, in coordination with NASA, a sufficiently large set of project data to use as a testbed for the Phase II effort.

Task 2: Hardware and Software Acquisition

- Install equipment and software — All necessary computer hardware and software will be purchased, configured, and installed at Analytics' offices.

Task 3: Detailed System Design

- Design and prototype estimation algorithms — The five algorithms (three **analog selection,** two **estimate from analog(s)**) will be designed and prototyped with the data base obtained during Task 1.
- Design project data base — The project data base will be designed based on information obtained during Task 1.
- Design project knowledge base — The project knowledge base will be designed based on information obtained during Task 1.
- Design and prototype user interface — The user interface will be designed and prototyped.
- Design and prototype intelligent dialogue mechanism — The intelligent dialogue mechanism will be designed and prototyped.
- Design and prototype user profiling mechanism — The mechanism for the user profile will be designed and prototyped.
- Finalize the design of all components of the INSITE system and determine how they will be integrated.

Task 4: System Development

- Implement INSITE control mechanisms — The high-level system shell will be implemented on the target system.
- Implement estimation algorithms — The five estimation algorithms designed and prototyped in Task 3 will be implemented in the delivery environment.
- Implement project data base — The project data base which was designed during Task 3 will be implemented and populated.
- Implement project knowledge base — The project knowledge base which was designed during Task 3 will be implemented and populated.
- Implement user interface — The user interface prototyped and validated during Task 3 will be implemented in the delivery environment.
- Implement intelligent dialogue mechanism — The intelligent dialogue mechanism prototyped during Task 3 will be implemented in the delivery environment.
- Implement user profiling mechanism — The user profiling mechanism prototyped during Task 3 will be implemented in the delivery environment.

Task 5: System Evaluation

- Refine estimation algorithms — The five estimation algorithms prototyped in Task 3 will be refined using data obtained during Task 1.
- Refine knowledge base — The knowledge base of project characteristics and their relationships will be refined using data obtained during Task 1.
- Refine user interface — Opinions on the user interface prototype will be solicited from the expert panel and NASA personnel and their suggestions incorporated into the final design.

- Refine intelligent dialogue system — Opinions on the intelligent dialogue system prototype will be solicited from the expert panel and NASA personnel and their suggestions incorporated into the final design.
- Refine user profiling system — During the review of the user interface and intelligent dialogue system, the user profiling system will be analyzed and refined as necessary.

Task 6: System Documentation

- User's Guide — A user's guide will be developed for use with the Phase II version of INSITE.
- Phase II Report — A report of progress and a discussion of future development will be written.

8.4 SUMMARY

A set of system design criteria was established following the Phase I interviews with NASA staff and a survey of the current technologies available. The proposed INSITE system is based on both proven and new technology areas. Four primary system components comprise the INSITE concept:

1. The estimation module;
2. The system manager;
3. The knowledge/data base; and
4. The user interface.

A partial proof-of-concept experiment was performed which successfully demonstrated use of statistical selection of analogs and the inference of estimates via central tendency among analogs.

The results of the Phase I research effort indicate the technical feasibility of the INSITE system. The obvious utility of an accurate and easy-to-use time estimation system warrants the development of a prototype INSITE system during a Phase II effort.

The INSITE system will be optimized continuously during the Phase II development. The two primary goals which will drive the optimization are:

1. The ability of the INSITE system to supply accurate estimates for project activities; and
2. Ease of user-computer interaction.

The first goal must be achieved in order to provide a useful answer to the user. The second goal is also important in order to obtain operator acceptability of the INSITE system. In summary, the INSITE system provides for enhanced project management capabilities to meet the goals of future NASA missions.

BIBLIOGRAPHY

Alavi, M. (1984). An assessment of the prototyping approach to information systems development. Communications of the ACM, 27(6), pp. 556-563.

Barto, A., and Anandan, P. (1985). Pattern-recognizing stochastic learning automata. IEEE Transactions on Systems, Man, and Cybernetics, 15(3), pp. 360-375.

Bennett, J. (1984). Managing to Meet Usability Requirements: Establishing and Meeting Software Development Goals. In Visual Display Terminals, J. Bennett, D. Case, J. Sandelin, and M. Smith (Eds.). Englewood Cliffs, NJ: Prentice-Hall, Inc., pp. 161-184.

Biswas, G., Oliff, M., and Sen, A. (1985). Design of an expert system in operations analysis. IEEE Computer, pp. 121-125.

Brachman, R. (1983). What IS-A is and isn't: An analysis of taxonomic links in semantic networks. IEEE Computer, 16(10), pp. 30-36.

Brachman, R. (1985). "I lied about the trees" or defaults and definitions in knowledge representation. The AI Magazine, Fall, pp. 80-92.

Brachman, R., and Levessque, H. (Eds.) (1985). Readings in Knowledge Representation. Los Altos, CA: Morgan and Kaufmann Publishers.

Buchana, B., and Shortliffe, E. (1985). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, MA: Addison-Wesley.

Buede, D., Yates, G., and Weaver, C. (1985). Concept design of a program manager's decision support system. IEEE Transactions on Systems, Man, and Cybernetics, pp.15(4), pp. 457-468.

Buffalano, C., Fogleman, S., and Gielecki, M. (1976). Cost Estimation: An Expert-Opinion Approach. NASA Technical Note TN D-8256. Washington, DC: NASA.

Carbonell, J. (1983). Learning by Analogy: Formulating and Generalizing Plans from Past Experience. Machine Learning: An Artificial Intelligence Approach. Palo Alto, CA: Tioga Publishing Co., pp. 137-161.

Charniak, E., and McDermott, D. (1985). Introduction to Artificial Intelligence. Reading, MA: Addison-Wesley.

Chouraqui, E. (1985). Construction of a Model for Reasoning by Analogy. In Progress in Artificial Intelligence, L. Steels and A. Campbell (Eds.). Chichester, England: Ellis Horwood Limited.

Cleland, D.L., and King, W.R. (1983). Project Management Handbook. New York: Van Nostrand Reinhold Co., Inc.

Clocksin, W., and Mellish, C. (1984). Programming in Prolog, 2nd Edition. Springer-Verlag.

Cohen, P., and Feigenbaum, E. (1982). The Handbook of Artificial Intelligence, Volume III. Stanford, CA: Heuris Tech Press.

Copi, I. (1954). Symbolic Logic. New York: MacMillan.

Defense Systems Management College (1983). System Engineering Management Guide. Fort Belvoir, VA: Defense Systems Management College.

Defense Systems Management College (1984). Department of Defense Manufacturing Management Handbook for Program Managers. Fort Belvoir, VA: Defense Systems Management College.

Eliot, L. (1986). Analogical problem solving and expert systems. IEEE Expert, Summer, pp. 17-26.

Evans, T. (1968). A Heuristic Program to Solve Geometric Analogy Problems. In Semantic Information Processing, M. Minsky (Ed.). Cambridge, MA: MIT Press.

Findler, N., Lo, R., and Bhaskaran, P. Two theoretical issues concerning expert systems. IEEE Computer, pp. 236-240.

Finin, T. (1986). Part II: Understanding frame languages implementing PFL. AI Expert, December, pp. 51-56.

Fischler, M., and Firschein, O. (1986). Intelligence: The Eye, the Brain, and the Computer. Reading, MA: Addison-Wesley.

Fox, M. (1982). The Intelligent Management System: An Overview. Processes and Tools for Decision Support. Netherlands: North-Holland Co.

Fox, M., Roth, S., Sathi, A., and Mattis, J. (1986). Callisto: An intelligent project management system. The AI Magazine, Winter, pp. 34-52.

Freiling, M., Alexander, J., Messick, S., Rehfuss, S., and Shulman, S. (1985). Steps towards automating expert system development. IEEE Computer, pp. 985-993.

Greenberg, S., and Witten, I. (1985). Adaptive personalized interfaces — A question of viability. Behaviour and Information Technology, 4, pp. 31-45.

Haack, S. (1978). Philosophy of Logics. Cambridge, England: Syndics of the Cambridge University Press.

Hankins, G., Jordan, J., Katz, J., and Mulvehill, A. (1985). Expert Mission Planning and Replanning Scheduling System. Technical Report M85-33. Bedford, MA: The MITRE Corporation.

Hartzband, D., and Maryanski, F. (1985). Enhancing knowledge representation in engineering data bases. IEEE Computer, September, pp. 39-48.

Hillier, F., and Lieberman, G. (1974). Operations Research, 2nd Edition. San Francisco, CA: Holden-Day, Inc.

Ihara, J. (1987). Extension of conditional probability and measures of belief and disbelief in a hypothesis based on uncertain evidence. IEEE Transactions on Pattern Analysis and Machine Learning, 9(4), pp. 561-568.

Kahane, H. (1973). Logic and Philosophy. Belmont, CA: Wadsworth Publishing Co.

Kemerer, C. (1987). An empirical validation of software cost estimation models. Communications of the ACM, 30(5).

Klaus, B., and Horn, P. (1986). Robot Vision. Cambridge, MA: Wadsworth Publishing Co.

Kumara, S., and Kashyap, R. Knowledge representation in expert systems via entity-relationships and its applications. IEEE Computer, pp. 495-500.

Lecot, K., and Parker, D. (1986). Control over inexact reasoning. AI Expert, Premier, pp. 32-43.

Lipschutz, S. (1965). Probability. New York: McGraw-Hill.

Loveland, D.W. (1984). Knowledge Acquisition and Evaluation within Expert Systems. Defense Technical Information Center Technical Report ADP003035. Alexandria, VA: Defense Logistics Agency, DTIC.

Malone, T., Grant, K., Turbak, F., Brobst, S., and Cohen, M. (1987). Intelligent information-sharing systems. Communications of the ACM, 30(5), pp. 390-402.

Mandi, A., and Chu, Y. (1985). Network model-guided expert knowledge elicitation. IEEE Computer, pp. 10-13.

Maron, M., Curry, S., and Thompson, P. (1986). An inductive search system: Theory, design, and implementation. IEEE Transactions on Systems, Man, and Cybernetics, 16(1), pp. 21-28.

Martin, J. (1973). Design of Man-Computer Dialogues. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Mason, R., and Carey, T. (1983). Prototyping interactive information systems. Communications of the ACM, 26(5), pp. 347-354.

Michalski, R., and Stepp, R. (1983). Automated construction of classifications: Conceptual clustering versus numerical taxonomy. IEEE Transactions on Pattern Analysis and Machine Intelligence, 5(4).

Moore, F., and Hendrick, T. (1977). Production/Operations Management, 7th Edition. Homewood, IL: Richard D. Irwin, Inc.

Nakamura, K., and Iwai, S. (1982). Topological fuzzy sets as a quantitative description of analogical inference and its application to question-answering systems for information retrieval. IEEE Transactions on Systems, Man, and Cybernetics, 12(2), pp. 558-568.

Nakamura, K., Sage, A., and Iwai, S. (1983). An intelligent data base interface using psychological similarity between data. IEEE Transactions on Systems, Man, and Cybernetics, 13(4), pp. 193-204.

Negoita, C. (1985). Expert Systems and Fuzzy Systems. Menlo Park, CA: The Benjamin/Cummings Publishing Co., Inc.

Norman, D., and Draper, S. (Eds.) (1986). User-Centered System Design: New Perspectives on Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Oppenheimer, J. (1956). Analogy in science. American Psychologist, 11.

Otto, H. (1978). The Linguistic Basis of Logic Translation. Washington, DC: University Press of America.

Pitrat, J. (1984). An Intelligent System Can and Must Use Declarative Knowledge Efficiently. In Artificial and Human Intelligence, A. Elithorn and R. Banerji (Eds.). Elsevier Science Publisher, pp. 271-280.

PRC Government Information Systems (1985). A Bibliography for NASA Managers. Washington, DC: NASA Science and Technical Information Branch.

Prerau, D. (1985). Selection of an appropriate domain for an expert system. The AI Magazine, Summer, pp. 26-36.

Rada, R. (1985). Gradualness facilitates knowledge refinement. IEEE Transactions on Pattern Analysis and Machine Intelligence, 7(5), pp. 523-530.

Rasmussen, J. (1985). The role of hierarchical knowledge representation in decision making and system management. IEEE Transactions on Systems, Man, and Cybernetics, 15(2), pp. 234-243.

Rauch, H. (1984). Probability concepts for an expert system used for data fusion. The AI Magazine, Fall.

Reitman, W. (1965). Cognition and Thought. New York: Wiley.

Rine, D. (1985). Some applications of fuzzy logic to future and expert systems. IEEE Computer, pp. 351-356.

Rumelhart, D., and Abrahamson, A. (1973). A model for analogical reasoning. Cognitive Psychology, 5.

Rushinek, A., and Rushinek, S. (1986). What makes users happy? Communications of the ACM, 29.

Sathi, A., Fox, M., and Greenberg, M. Representation of activity knowledge for project management. IEEE Transactions on Pattern Analysis and Machine Intelligence, 7(5), pp. 531-552.

Schank, R. (1984). Memory-Based Expert Systems. Defense Technical Information Center Technical Report AFOSR-TR-84-0814. Alexandria, VA: Defense Logistics Agency, DTIC.

Schnirring, B. (1986). Artificial Intelligence. NASA Tech Briefs, Special Edition, pp. 24-30.

Shen, H. and Chan, K. (1985). An Aid for Knowledge Acquisition in Knowledge-Based Systems, Pattern Analysis and Machine Intelligence Group. Department of Systems Design Engineering, University of Waterloo, Waterloo, Canada.

Shneiderman, B. (1986). Design the User Interface: Strategies for Effective Human-Computer Interaction. Reading, MA: Addison-Wesley.

Silverman, B. (1983). Analogy in systems management: A theoretical inquiry. IEEE Transactions on Systems, Man, and Cybernetics, 13(6), pp. 1049-1075.

Silverman, B. (1985). The use of analogs in the innovation process: A software engineering protocol analysis. IEEE Transactions on Systems, Man, and Cybernetics, 15(1), pp. 30-44.

Silverman, B., and Moustakis, V. (1987). Expert Systems Issue in INNOVATOR: Representations and Heuristics. In Expert Systems for Business, B. Silverman (Ed.). Reading, MA: Addison-Wesley, pp. 402-439.

Smith, C., Irby, C., Kimball, R., Verplan, B., and Harslem, E. (1982). Design the Star User Interface. Byte, 7(4), pp. 242-282.

Spearman, C. (1927). The Abilities of Man. New York: Macmillan.

Starr, M. (1964). Production Management: Systems and Synthesis. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Sternberg, R. (1977). Component processes in analogical reasoning. Psychological Review, 84, pp. 353-378.

Strawson, P.F. (1966). Introduction to Logical Theory. London: Methuen.

Trevellyan, R., and Browne, D. (1987). A Self-Regulating Adaptive System. In Proceedings of the 1987 Human Factors in Computing Systems and Graphics Interface, pp. 103-107.

Trig, C. (1973). Guidelines for Cost Estimation by Analogy. Research and Development Technical Report ECOM-4125. Fort Monmouth, NJ: United States Army Electronics Command.

Vere, S. (1983). Planning in time: Windows and durations for activities and goals. IEEE Transactions on Pattern Analysis and Machine Intelligence, 5(3), pp. 246-266.

Waterman, D. (1986). A Guide to Expert Systems. Reading, MA: Addison-Wesley.

Wescourt, K., and Thorndyke, P. (1983). Alternative Knowledge Acquisition Interface Structures. Defense Technical Information Center Technical Report PPAFTR-1131-83-1. Alexandria, VA: Defense Logistics Agency, DTIC.

Williamson, M. (1986). Project costing with COCOMO1. AI Expert, November, pp. 52-57.

Winston, H. (1984). Artificial Intelligence. Reading, MA: Addison-Wesley.

Woods, W. (1975). What's in a link: Foundations for semantic networks. Representation and Understanding: Studies in Cognitive Sciences, D. Bobrow and A. Collins (Eds.). New York: Academic Press, pp. 35-82.

Yaghmai, N., and Maxin, J. Expert systems: A tutorial. Journal of the American Society for Information Science, 35(5), pp. 297-305.

Zadeh, L.A. (1984). Making computers think like people. IEEE Spectrum, August, pp. 26-32.

Zadeh, L.A. (1985). Syllogistic reasoning in fuzzy logic and its application to usuality and reasoning with dispositions. IEEE Transactions on Systems, Man, and Cybernetics, 15(6), November/December.

®