DECISION
SCIENCE
CONSORTIUM, INC.

TESTING AND EVALUATING C^{3}I SYSTEMS THAT EMPLOY AI

(CLIN 0001)

VOLUME 1: HANDBOOK FOR TESTING EXPERT SYSTEMS

Leonard Adelman, Jacob W. Ulvila, and Paul E. Lehner

Decision Science Consortium, Inc.
1895 Preston White Drive, Suite 300
Reston, Virginia 22091

January 1991

Final Report
Period of Performance: 16 September 1988 - 15 September 1990
Contract Number: DAEA18-88-C-0028
PR&C Number: W61DD3-8057-0601
AAP Number: EPG 8048

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Prepared for:
U.S. Army Electronic Proving Ground
ATTN: STEEP-ET-S (Mr. Robert J. Harder)
Fort Huachuca, Arizona 85613-7110

The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

TECHNICAL REPORT 90-9

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): 90-9	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Decision Science Consortium, Inc. / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: US Army Electronic Proving Ground STEEP-ET-S
6c. ADDRESS (City, State, and ZIP Code): 1895 Preston White Drive, Suite 300 Reston, Virginia 22091	7b. ADDRESS (City, State, and ZIP Code): Ft. Huachuca, Arizona 85613-7110
8a. NAME OF FUNDING/SPONSORING ORGANIZATION / 8b. OFFICE SYMBOL (If applicable): STEEP-ET-S	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAEA-18-88-C-0028
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification): TESTING AND EVALUATING C^3I SYSTEMS THAT EMPLOY AI -- VOLUME 1: HANDBOOK FOR TESTING EXPERT SYSTEMS

12. PERSONAL AUTHOR(S): Leonard Adelman, Jacob W. Ulvila, and Paul E. Lehner

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Technical	FROM Sep 88 TO Sep 90	1991 January 31	334

16. SUPPLEMENTARY NOTATION: The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Expert Systems, Testing, Knowledge-Based Systems, Artificial Intelligence, Multiattribute Utility

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This is the first and main volume of a five-volume report on this project to develop methods for testing expert systems. This volume provides a software tester with a comprehensive method for testing expert systems and knowledge-based systems. It contains chapters on an overview of expert system testing, foundations for testing expert systems, subjective methods, technical methods, empirical methods, an integrative framework for testing and evaluation, the relationship between this framework and other approaches to testing, and future directions. It also contains a detailed questionnaire that can be used to elicit subjective information from subjects and an extensive list of references.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Mr. Robert J. Harder	22b. TELEPHONE (Include Area Code): (602) 538-2090 / 22c. OFFICE SYMBOL: STEEP-ET-S

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE: UNCLASSIFIED

PREFACE

This report describes the results of a research project funded under the Small Business Innovative Research (SBIR) program by the U.S. Army Electronic Proving Ground. Phase 1 of this project was conducted from September 1986 to March 1987. It resulted in a report and a prototype software program, TESTER. Phase 2 was conducted from September 1988 to September 1990 and resulted in a five-volume report and a software prototype, TESTER_C. Volume 1 of the report *Handbook for Testing Expert Systems*, provides a comprehensive approach to testing expert systems. Volume 2, *Compendium of Lessons Learned from Testing AI Systems in the Army*, provides the results of a survey of software testers and offers suggestions for improving the practice of testing AI software. Volume 3, *A Guide to Developing Small Expert Systems*, provides a step-by-step guide for a beginner. Volume 3 was also delivered in a hypertext version. Volume 4, *Published Articles*, contains copies of the six published articles developed in this project. Volume 5, *User's Manual for TESTER_C*, is a user's manual for the prototype software that implements the multiattribute utility analysis (MAUA) framework for testing and evaluating expert systems described in Volume 1.

Volume 1 is intended as a handbook that can be used by a tester interested in testing a knowledge-based system or an expert system. The reader interested in an overview of our methods may wish to skim Chapter 7, "Pulling It All Together," first. He or she may then wish to review the attributes in the MAUA hierarchy presented in the latter half of Chapter 3, "Proposed MAUA Framework for Testing and Evaluating Expert Systems." A detailed example of the method is given in Volume 5, *User's Manual for TESTER_C*. After this introductory review, a tester should read Chapters 1 and 2 to gain an overview of the expert system testing methods. The reader may then wish to pick and choose among the specific methods described in Chapters 3, 4, 5, and 6 to gain more in-depth knowledge of the techniques needed for the particular test. Chapter 8 contrasts the methods of this Handbook with other approaches to software testing and evaluation. Finally, Chapter 9 describes areas where further research and development are needed.

As with many monographs on software, this one mentions certain products. EXSYS is a trademark of Exsys, Inc. CLIPS is a product of NASA. Other product names used in this document may be trademarks of their respective companies.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

Page

TABLES

FIGURES

CONTENTS (Continued)

Page

FIGURES

CONTENTS (Continued)

Page

FIGURES

CHAPTER 1:

TESTING AND EVALUATING EXPERT SYSTEMS: AN OVERVIEW AND ILLUSTRATION

The test and evaluation of expert systems is becoming increasingly important, for expert systems are moving out of the laboratory and into operational use. How good are these systems? Do they do what their developers claim? Do they meet the users' requirements? Are their knowledge bases reliable and valid? Do they actually improve operator (and organizational) performance? Can they be effectively integrated with and maintained among more conventional software systems? Test and evaluation methods provide a means of answering these and many other questions for the user community. More generally, test and evaluation provides the feedback required for keeping the expert system development process on track and, thereby, increasing the probability that the expert system will be used and effective.

Expert system technology holds great promise for many reasons. First, the financial cost to build expert systems has gone down. Expert system software (e.g., shells) is now much more affordable than it was just five years ago. Moreover, many shells are now available on personal computers, thereby decreasing the implementation costs and problems that existed when one needed expensive, expert system hardware. In addition, we now have a much better idea of how to build expert systems. Our experience in building expert systems has gone up significantly in a very short time. [Note that we use the term "expert system" generally, to include all classes of knowledge-based systems.]

Second, we have some clear commercial successes to point to; successes other than MYCIN, PROSPECTOR, XCON, or other pioneering systems. To illustrate this point, we can point to expert systems actually helping to (1) process loan applications for Citibank (Keyes, 1989), (2) monitor the safety of mines in the U.S. and other countries, (Newquist, 1988), (3) process insurance claims for Blue Cross/Blue Shield of South Carolina (Weitzel and Kerschberg, 1989), (4) diagnose functional problems with robots at Ford Motor Company (Smith, 1988), (5) monitor the performance of on-line networks at Sumitomo Metal Industries of Japan (Newquist, 1988), etc.

Third, indications are that expert systems have barely impacted their potential market. To quote Wolfgram et al. (1987, p. 21), "Many industry analysts estimate that currently only 10% of potential expert system applications are being recognized ..." And even after 1988, which Chapnick (1988, p. 5) indicated "won't be considered a banner one for AI ..., everyone is still predicting relatively high compounded growth rates (greater than 30%) through the mid-1990s." Moreover, this estimate does not appear to include mass-market expert systems applications, which combine expert-systems technology with traditional applications-oriented software for the mass market. Examples of currently available mass-market expert systems include AskDan for tax preparation, SELLSTAR for sales-tracking and advice, Ex-Sample for determining the appropriate sample size for a research project, and STS/Expert for stocks. To quote Eliot (1989, p. 9), "Mass-market applications are the future of the expert-systems industry and will affect applications everywhere."

Although there have been successes, there also have been failures. In fact, many expert systems that are developed are simply not used. To quote Casey (1989, p. 44), "For every success story, however, many expert-system development projects have failed or are in deep trouble. Many expert systems end up either 'dead on arrival' (never work), among the ranks of the unemployed (never used), or serving a life sentence in research and development (never finished)." The Department of Defense, for example, has spent millions of dollars on expert system technology with minimal transfer to operational personnel. And private industry has spent millions of dollars developing expert systems with minimal impact on the size of the workforce these expert systems were to replace.

The reasons for this state of affairs lie, of course, on both sides of the fence. As Andriole (1989, p. 7) points out when discussing all forms of decision support systems technology, "Vendors have vested interests in overselling, and users are inclined to want to believe that a solution to all their problems can be found on one or two floppy disks." However, a focus on motives obscures the bigger issues. For as Andriole (p. 7) points outs, "The truth of the matter is that the state of the art of decision support systems technology is unbalanced and evolving."

These statements are just as true for expert systems technology. What has been unbalanced is that, all the rhetoric to the contrary, expert system development efforts have until recently been primarily technology-driven. What is evolving is a more requirements-driven, expert systems development process. The requirements-driven evolution is, of course, taking many forms. Three aspects of it are overviewed here. When considering them, the reader should keep the concept of "balance" in mind.

First, there is a growing realization that the success of an expert system development effort depends on picking the right problem. To quote Casey (1989, p. 44), "One simple rule for success that all would-be developers should repeat out loud each day during morning calisthenics is, 'Pick the right problem.' Just as location is the biggest factor in real estate, selecting the right problem is absolutely essential to expert system development. Unfortunately, the importance of selecting the right application is often lost in the excitement and enthusiasm accompanying the initial decision to use expert-system technology." Nor is it an easy problem. Casey has even built an expert system called ESES (Expert System Expert System) to help in problem selection.

Second, the users' needs are an essential aspect of problem selection. The concern with "picking the right problem" is, of course, not new. In the past, however, it has primarily focused on the characteristics of the task and the experts who would provide the knowledge. Does the task primarily require symbolic reasoning? Does it require the use of heuristics? Are decisions based on incomplete and/or uncertain information? Is there a knowledge czar or are there high levels of agreement among experts? Are experts available over a long period of time?

More recently, the questions have also begun emphasizing the potential user's explicit needs. To quote Smith (1988, p. 53), "In addition, you should try to choose applications that: Are real. Don't try to solve problems that don't exist—you'll only create systems nobody will use. Fit in with your organization's future direction and plans ... Have measurable benefits. Pay particular attention to such things as cost reductions and improvements in quality, productivity, and working conditions. However, don't overlook

intangible benefits. These are sometimes referred to as 'warm fuzzies,' because even though you can't quantify them, they are nice to have."

Third, test and evaluation are essential to keeping the expert system development effort focused on users' needs. Again, one might say, this isn't news. After all, isn't the purpose of prototyping to develop an illustrative system so that potential users can evaluate it? So that they can make sure that the final product meets their needs?

Unfortunately, prototyping has not been as successful as we might like to believe. In an editorial in *AI Expert*, Chapnick (1988, p. 5) referred to "[t]he more general problem of the lamentable, unimplementable prototype ..." And Cholawsky (1988, p. 42) points out that "[t]he inability to move from a prototype effort to an operational, delivered system is a chronic problem for organizations developing expert systems." In an effort to help cure the "prototyping blues," she emphasizes the importance of prototype planning that explicitly identifies objectives and evaluation criteria for determining prototype success. Nor is she alone. Adelman and Ulvila (in press), Andriole (1989), and much earlier, Gaschnig et al. (1983) have all argued for the importance of specifying explicit test and evaluation criteria early in the prototype development process in order to keep development on track. Yet a recent survey by Constantine and Ulvila (in press) has found that such criteria or, more generally, requirements, are not specified in many (if not most) expert system development efforts.

Concurrent with the evolution of a more requirements-driven development process has been the evolution of methods for testing expert systems against evaluation criteria. The American Association for Artificial Intelligence (AAAI) held its first workshop on test and evaluation methods in Minneapolis, MN, in August, 1988. This, the 1989 IJCAI workshop, and the 1990 AAAI workshop focused primarily on methods for assessing the logical consistency and completeness of the knowledge base (AAAI Workshop Proceedings, 1988; IJCAI-89 Workshop Proceedings, 1989). There is a growing awareness, however, that testing and evaluation is multi-faceted. As experience in software testing (e.g., see Beizer, 1984; Hamlet, 1988) has shown, no single method is completely adequate. In the case of expert systems, one must also consider

methods for assessing (1) the subjective opinion of users (e.g., Adelman and Donnell, 1986; Klein and Brezovic, 1988; Ulvila et al., 1987); (2) the predictive accuracy of the knowledge base (e.g., see Lehner, 1989; Lehner and Ulvila, 1989; O'Keefe et al., 1987); and (3) the overall performance of the organization using the system (e.g., Adelman, 1990a,b; Adelman and Ulvila, in press).

The purpose of this book is to show one how to perform formal tests and evaluations of expert system technology. It is a methods book. The goal is to provide one with an understanding of the procedures required to perform effective tests and evaluations, and how to incorporate these procedures into the expert system development process. Moreover, to the extent possible, this book provides illustrative examples of how to utilize formal test and evaluation procedures to help readers to apply these procedures to ongoing expert system developments. In short, the orientation is to provide a step-by-step description of how to test and evaluate expert systems. This volume is intended primarily for major expert systems. Volume 3 of this report provides guidance for small expert systems.

It must be emphasized at the outset, however, that no methods book on test and evaluation can be a "cookbook" because the focus of test and evaluation is to ensure that the technology being developed is consistent with the user's requirements. Unfortunately, users are often uncertain of exactly what their decision requirements are, requirements analysis techniques are more than fallible, and the procedures for converting requirements analyses to system functions are still being refined by researchers. As a result, the development team is faced with numerous judgments and decisions. Indeed, it is the pervasiveness of these judgments and decisions that make successful expert system development so difficult.

Broadly speaking, test and evaluation methods are tools for structuring and making the judgments and decisions inherent in the system development process. As such, they represent the control mechanism for finding out what needs to be done to increase the probability that the expert system will be used by the decision maker(s) for whom the system is being built and, in turn, improve organizational decision making and performance. Because evaluation

serves as a control mechanism for the development process, readers also need a broad framework for considering evaluation issues, as well as specific test and evaluation methods, in order to keep the development process on track. This book will provide readers with such a framework. Moreover, the book will show readers how the broad framework and specific methods can be integrated into the development process.

Thus far, we have not distinguished between "test" and "evaluation." We will do so here. Specifically, we will use the term "test" to refer to the process of measuring the expert system's performance against specific criteria. These criteria are generally referred to as "measures of effectiveness" (MOEs). The measurement approach may be (1) logically-based, such as testing the logical consistency of the rules in the knowledge base; (2) empirically-based, such as testing the predictive accuracy of the knowledge base against the judgmental accuracy of experts or ground-truth measures of accuracy; (3) observationally-based, such as recording the features of the expert system that users routinely use when solving test cases; or (4) subjectively-based, such as using questionnaires to assess users' opinions of the system's strengths and weakness.

We will use the term "evaluation" to refer to the process of aggregating all the different tests in order to reach an overall conclusion about the expert system. Central to the concept of "evaluation" is the concept of "relative importance weights," or alternative decision rules, for combining good test scores on some MOEs with bad test scores on others. Relative importance weights represent personal judgments. We will argue from the outset that such judgments should be made by the decision makers, or their representatives, who are sponsoring the development of the expert system—not by the testers. This initially might be disturbing to, and difficult for, members of the sponsoring, development, and evaluation teams, for it emphasizes the subjective process decision makers go through when evaluating the overall value of an expert system. However, it is quite consistent with a requirements-driven development approach. Moreover, to quote Riedel and Pitz (1986, pp. 987-988), "There is no way to avoid the fact that the overall MOE must be based on such judgments, or the fact that no mechanical procedure can replace this subjective assessment ..."

The remainder of the first chapter is divided into three sections. The first section presents a decision-making perspective and paradigm as a backdrop for considering the decision to develop an expert system, and the role of test and evaluation with respect to this decision. The second section overviews a multi-faceted, test and evaluation approach for providing the range of information required by the organization building the expert system. The third section provides a case study showing how this approach can provide this information and, thereby, enhance prototype development.

A DECISION-MAKING PERSPECTIVE AND PARADIGM

When testing and evaluating expert systems, it is important to remember the obvious, which is that the overall aim of an expert system is to improve the effectiveness of the organization using it. Improved organizational effectiveness can occur in many ways, such as through decreased personnel costs, greater access to expert knowledge, or improved decision making. The latter focus will be emphasized throughout this book because of the ever increasing importance given to effective decision making for the success of post-industrial organizations (e.g., see Huber, 1986).

Simon (1960) has used three categories to describe decision-making activities: intelligence, design, and choice. "Intelligence" refers to the activities inherent in problem identification, definition, and diagnosis. It is, as Huber (1980) points out, the conscious process of trying to explore the problem in an effort to find out the current state of affairs, and why it does not match our desires. "Design" refers to those activities inherent in generating alternative solutions or options for solving the problem. It involves "... identifying items or actions that could reduce or eliminate the difference between the actual situation and the desired situation" (Huber, 1980, p. 15). And "choice" refers to those activities inherent in evaluating and selecting from the alternatives. It is the action that most people think of when one makes a decision.

As Huber (1980) and others (e.g., Andriole, 1989; Sage, 1986; Wohl, 1981) have pointed out, decision-making activities are a subset of problem-solving activities. For example, the first three steps in Huber's five-step

problem-solving paradigm are those activities that require (1) problem identification, definition, and diagnosis; (2) the generation of alternative solutions; and (3) evaluation and choice among alternative solutions. These steps are conceptually identical to Simon's decision-making categories. The fourth step in Huber's paradigm involves those activities inherent in implementing the chosen alternative. The fifth step involves those activities inherent in reviewing or monitoring the implemented action in an effort "...to see that what actually happens is what was intended to happen" (Huber, 1980, p. 19). If there is a significant mismatch between the actual and desired state of affairs, we are back to step #1, exploring the problem.

Although it is presented within the context of military tactical decision making (and aiding), Wohl (1981) has presented a problem-solving paradigm that explicitly identifies the evaluation functions inherent in decision making. Figure 1-1 presents Wohl's (1981, p. 625) SHOR (Stimulus-Hypothesis-Option-Response) paradigm. Intelligence activities are differentiated between the Stimulus and Hypothesis elements of the SHOR paradigm. In particular, the Stimulus element includes data collection, correlation, aggregation, and recall activities; it naturally includes many of the activities also included in Huber's last problem-solving stage—that of monitoring the situation. The Hypothesis element is that aspect of Intelligence that involves creating alternative hypotheses to explain the cause(s) of the problem, evaluating the adequacy of each hypothesis, and selecting one hypothesis as the most likely cause of the data.

On the basis of the selected hypothesis, or hypotheses if one cannot differentiate between hypotheses because of the uncertainty and/or ambiguity in the data, the decision maker generates alternative options for solving the problem. As in Simon's and Huber's paradigms, the Option element in the SHOR paradigm explicitly differentiates between option creation, evaluation, and selection activities. Finally, on the basis of the selected option, the decision maker takes action, which includes the planning, organization, and execution of a Response to the problem, analogous to the fourth step in Huber's problem-solving framework.

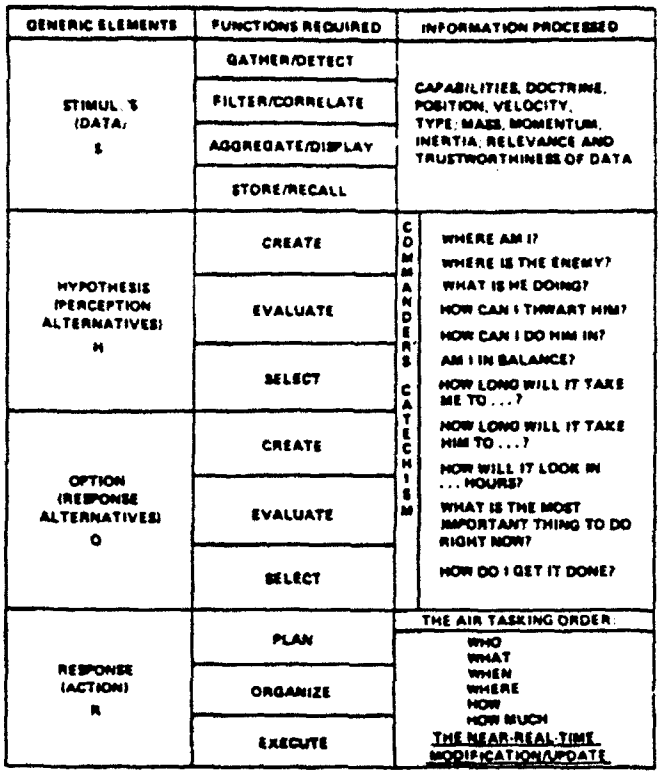


Figure 1-1: The SHOR Paradigm
(from Wohl, 1981; last column for illustrative purposes only)

As Wohl (1981, p. 626) points out, the "... SHOR paradigm is basically an extension of the stimulus response (SR) paradigm of classical behaviorist psychology to provide explicitly for the necessity to deal with two realms of uncertainty in the decision-making process: (1) information input uncertainty, which creates the need for hypothesis generation and evaluation; and (2) consequence-of-action uncertainty, which creates the need for option generation and evaluation." Different elements of the SHOR paradigm become more or less important depending on where the uncertainty resides. For example, "Where options are more or less clearly prescribed but input data is of low quality (e.g., as in military intelligence analysis), a premium is placed upon creation and testing of hypotheses (e.g., where is the enemy and what is he doing?). Where input data are of high quality but options are open-ended (e.g., as in the Cuban missile crisis), a premium is placed upon creation and analysis of options and their potential consequences (e.g., if we bomb the

missile sites or if we establish a full-fledged naval blockade, what will the Russians do ?) ... By contrast, tactical decision-making in support of combined air-land operations is generally characterized by both poor quality input data and open-ended options; hence, there is a much greater need than in other military situations for rapid hypothesis and option processing in the field" (Wohl, 1981, p. 626).

As Adelman (1987) has pointed out, the SHOR paradigm also is consistent with more currently popular cognitively oriented paradigms. For example, the script theory representation by Shank and Abelson (1977), the schema theory representation by Noble (1989), and the fuzzy set decision rule representation by Zimmermann and Zysno (1980) all have both situation assessment and action components. The situation assessment component typically operates via a 'pattern matching' mechanism, which is consistent with the Stimulus and Hypothesis elements of the SHOR paradigm. Once a script or schema is activated, there is a set of actions that is consistent with it; this is consistent with the Option and Response elements of the SHOR paradigm.

When considering expert system test and evaluation, it is important to remember that the decision makers who have decided to build an expert system are in a tactical or strategic decision-making situation, depending on the forecasting and planning horizon under which they are operating. Moreover, the situation can be represented by the SHOR paradigm. For, on the basis of available and projected data, the decision makers are making hypotheses about the nature of the environment that they and their organization will face in the future. That is, they are forecasting the future state of affairs and trying to assess whether their current actions will be effective or not in achieving their future goals and objectives. And they are generating options to deal with their hypotheses regarding potential future performance shortfalls. Given all the stimuli about the dynamic nature of future business and government (particularly military) environments, the ever increasing role that decision making will play in organizational success, the decreasing financial cost of computer hardware, and the ever increasing power of computer systems to support decision making, it is not surprising that decision makers in many organizations think that expert system technology will be an effective response to their hypotheses about the future.

It is important for us to keep this "big picture" in mind when testing and evaluating expert systems. We must remember that hypotheses about the problem environment and judgments about the relative effectiveness (or utility) of various options are often made, respectively, under both information input and consequence-of-action uncertainty. It is important to realize that, at the time that it is made, the decision to develop an expert system is, in fact, nothing more than a hypothesis that this option will be an effective response to the problem environment. This may or may not be true. Other options, either singularly or in combination with the development of an expert system, may be better options. From this perspective, it can be argued that the ultimate goal of test and evaluation is to help senior-level decision makers in an organization decide whether the option of developing an expert system, either singularly or in combination with other actions, is an effective organizational response for dealing with the present and/or future problem environment.

Once the development process is underway, the application of formal test and evaluation methods permits one to monitor the perceived utility of the expert system under development and take corrective action to increase the probability of its use and effectiveness. This can be seen by using the SHOR paradigm to represent the expert system development process. Specifically, the development team's job is to plan, organize, and execute the selected option, which in this case is the development of a specific expert system. The purpose of test and evaluation is to systematically gather, filter, and aggregate data (i.e., stimuli) about the expert system under development in order to test the hypothesis that all is going well; that is, that the expert system will do what decision makers and users want it to do and, thereby, be valuable to them. If all is not going well, that is, if there is a problem or if it is not clear what action to take, then options need to be generated, evaluated, and selected for correcting the problem(s) so that the development process can be kept on track. This clearly requires iteration, and is quite consistent with a requirement-driven prototyping process.

As the above discussion implies, there are two groups of persons that utilize and, indeed, require the results of formal tests and evaluations. The first group is the development team. It is composed of user(s), designers,

knowledge engineers, domain experts, and programmers. The second group is the sponsoring team. If an expert system is being developed only for the use of a particular decision maker, then he or she is both the user and financial sponsor of the expert system. However, for many expert system development efforts, particularly those funded by the federal government, the sponsors and users of the expert system are distinctly different groups of people. As a result, "Policy decisions must be made about the system's design, implementation, fielding, funding, and incorporation into the organizational functioning. These decisions are made by program managers and sponsors and more general policymakers. The last group is usually interested in more general information about the aid's potential or actual effectiveness" (Riedel and Pitz, 1986, p. 984).

As Beizer (1984), Hetzel (1984), Riedel and Pitz (1986), and others have recommended, we will assume that the development team also includes testers and evaluators whose job is to obtain the test and evaluation data required to keep the development effort on track. For the simplicity of presentation, however, we will often use the terms "testers" and "evaluators" interchangeably. We realize that in many organizations, such as in the U.S. Army for example, "testers" and "evaluators" are distinctly different groups of trained individuals who would be found in different organizational units. We will try to maintain the distinction here too. However, we will at times blur the distinction to facilitate the presentation of material. We do not feel uncomfortable in doing so, because, from the perspective of this book, both "testers" and evaluators should be proficient in obtaining both test and evaluation data.

The sponsoring and development groups make different types of decisions during the expert system development process and, consequently, require different types of information upon which to base those decisions. Ideally, good tests and evaluations have to be capable of addressing the different needs of both groups. This requires the application of different test and evaluation methods, appropriately matched to the information and decision needs of different persons throughout the development process.

Unfortunately, it is often not possible to systematically incorporate all members of the sponsoring group into the expert system development process. (In fact, it is often difficult to get users to actively participate as members of the development team, although both research and common sense have demonstrated the importance of their participation to the successful implementation of all forms of decision support technology.) Numerous reasons are given for their lack of involvement, including busy schedules, a belief in "hands-off" policy during development, a lack of desire to be involved, a lack of money, etc. Evaluators need to be conscious of this problem and do what they can to incorporate members of the sponsoring team into the development process. Throughout the book we will discuss explicit evaluation methods for addressing the policymaking decisions about which members of the sponsoring team need information.

As the above discussion suggests, many expert system development efforts do not use explicit evaluation methods to provide a control mechanism for the development process. Obviously, we think that they should and, we will argue, that doing so will increase the probability of the successful implementation and value of the expert system. It is important to note that, as Riedel and Pitz (1986, p. 994) point out, "... user satisfaction with the aid is not a sufficient criterion for evaluation because of the extraneous factors that can affect satisfaction." There are numerous other factors, such as the quality of the decisions made with the expert system, the logical soundness, completeness and predictive accuracy of the knowledge base, the effectiveness of the match with personnel and organizational characteristics, etc., that go into making a good expert system. User satisfaction is, however, a necessary condition for use of the expert system. "... [I]n the final analysis, the purpose of developing an aid is to have it used, presuming it to be effective. Similarly, the purpose of the evaluation is to produce information that is used. This concern for impact on design or policy decisions is the determinant of what evaluation information to obtain. How to obtain that information in a valid manner is left to the expertise of the evaluator."

The evaluator's job is to select the method(s) that is most appropriate for the decision maker's questions, stage of the expert system development process, available funds, etc. The basic requirement is for an eclectic

approach that is based on the evaluation purpose and situation. The goal throughout is to provide guidance in making the judgments and decisions inherent in building the expert system. It is for this reason that evaluation has been referred to as the control mechanism that keeps the development process on track.

A MULTI-FACETED TEST AND EVALUATION APPROACH

Adelman and Donnell (1986) presented a three-phased (or faceted) approach for testing and evaluating decision support systems; Adelman and Ulvila (in press) recently extended it to expert systems and showed how it could be used when selecting classes of test and evaluation methods. The three-phase evaluation approach is composed of a subjective phase for obtaining users' opinions regarding the system's strengths and weaknesses; a technical evaluation phase for "looking inside the black box;" and an empirical evaluation phase for assessing the system's impact on performance. Specifically, the subjective evaluation phase focuses on evaluating the expert system from the perspective of potential users. The goal of the subjective evaluation is to assess whether the users like the expert system, what they consider to be its strengths and weaknesses, and what changes they would suggest for improving it.

The technical phase focuses on evaluating the expert system from both an internal (heuristic) perspective and an external (systemic input/output) perspective. For example, most people considering the technical evaluation of an expert system might focus on assessing the logical (and functional) adequacy and predictive accuracy of its knowledge base. Rushby (1988) has called these "competency requirements." However, from a transfer and maintenance perspective, one also needs to be concerned with conventional test and evaluation issues, such as whether the system can be effectively and efficiently integrated with other software and hardware systems in the operational environment, and whether it was designed consistent with the organization's design and coding standards. Rushby has called these concerns "service requirements." A comprehensive test and evaluation framework needs to address both classes of "technical" requirements.

The empirical evaluation phase focuses on obtaining objective measures of the system's performance. The goal of the empirical phase is to assess, for example, whether the system makes proper recommendations and whether persons make significantly better or faster decisions or use significantly more information working with, rather than without, the system, and to identify mechanisms for improving performance. It is important to note that the potential users of expert system technology may not be experts in the substantive domain. In these cases, one needs both experts and users to participate in the evaluation. The experts are needed for the technical evaluation of the knowledge base; the users for the empirical evaluation of system performance. If possible, experts should also participate in the empirical evaluation in order to systematically assess whether system performance is a function of user type. In addition, as will be illustrated in the case study presented later in this chapter, participation of domain experts in the empirical evaluation often provides insight into the functional completeness and predictive accuracy of the knowledge base.

For an evaluation to be effective, the evaluator must decide in advance what is to be tested. This is done by identifying measures of effectiveness (MOEs) that are designed to answer the evaluator's questions. These questions depend on who needs the information—that is, whether it is a member of the development or sponsoring team—the type of information needed, the stage of the development process, the interface being evaluated, etc. The resulting MOEs may be either logically-based, empirically-based, observationally-based or subjectively-based variables depending on the selected testing method, a point that will be returned to later in the chapter. The only restrictions are that each MOE must be measurable and that it provides the required information. Or to put it differently, the MOE must be correlated (positively or negatively) with the overall utility of the expert system under development.

Table 1-1 presents the hierarchy of subjective MOEs used in the case study presented later in this chapter. [Note: Chapter 2 presents a hierarchy of MOEs that (1) is more directed toward supporting the selection of test and evaluation methods, and (2) gives more emphasis to testing and evaluating the knowledge base.] The MOE hierarchy presented here was developed by Adelman

Table 1-1: Hierarchy of Measures of Effectiveness (MOEs) and Number of Questions Assessing Bottom-Level MOEs

0.0 OVERALL UTILITY (6)

1.0 DSS/USER INTERFACE

- 1.1 MATCH WITH PERSONNEL
  - 1.1.1 TRAINING/TECHNICAL BACKGROUND (3)
  - 1.1.2 WORKSTYLE/WORKLOAD/ INTEREST (4)
  - 1.1.3 OPERATIONAL NEEDS (5)
- 1.2 DSS' CHARACTERISTICS
  - 1.2.1 GENERAL
    - 1.2.1.1 EASE OF USE (4)
    - 1.2.1.2 UNDERSTANDING (3)
    - 1.2.1.3 EASE OF TRAINING (2)
    - 1.2.1.4 RESPONSE TIME (2)
  - 1.2.2 SPECIFIC
    - 1.2.2.1 USER INTERFACE (2)
    - 1.2.2.2 DATA FILES (3)
    - 1.2.2.3 EXPERT JUDGMENTS (2)
    - 1.2.2.4 ABILITY TO MODIFY JUDGMENTS (2)
    - 1.2.2.5 AUTOMATIC CALCULATIONS (2)
    - 1.2.2.6 GRAPHS (2)
    - 1.2.2.7 PRINTOUTS (2)
    - 1.2.2.8 TEXT (2)

2.0 USER-DSS/ORGANIZATION

- 2.1 EFFICIENCY FACTORS
  - 2.1.1 TIME
    - 2.1.1.1 TASK ACCOMPLISHMENT (3)
    - 2.1.1.2 DATA MANAGEMENT (2)
    - 2.1.1.3 SET-UP REQUIREMENTS (2)
  - 2.1.2 PERCEIVED RELIABILITY UNDER AVERAGE BATTLE CONDITIONS (2)
    - 2.1.2.1 SKILL AVAILABILITY (3)
    - 2.1.2.2 HARDWARE AVAILABILITY
- 2.2 MATCH WITH ORGANIZATIONAL FACTORS
  - 2.2.1 EFFECT ON ORGANIZATIONAL PROCEDURES AND STRUCTURE (2)
  - 2.2.2 EFFECT ON OTHER PEOPLE'S POSITION IN THE ORGANIZATION
    - 2.2.2.1 POLITICAL ACCEPTABILITY (2)
    - 2.2.2.2 OTHER PEOPLE'S WORKLOAD (2)
  - 2.2.3 EFFECT ON INFORMATION FLOW (2)
  - 2.2.4 SIDE EFFECTS
    - 2.2.4.1 VALUE IN PERFORMING OTHER TASKS (2)
    - 2.2.4.2 VALUE TO SAC OR OTHER SERVICES (2)
    - 2.2.4.3 TRAINING VALUE (2)

3.0 ORGANIZATION/ENVIRONMENT

- 3.1 DECISION ACCURACY (8)
- 3.2 MATCH BETWEEN DSS' TECHNICAL APPROACH AND PROBLEMS' REQUIREMENTS (7)
- 3.3 DECISION PROCESS QUALITY (2)
  - 3.3.1 QUALITY OF FRAMEWORK FOR INCORPORATING JUDGMENT (2)
  - 3.3.2 RANGE OF ALTERNATIVES (2)
  - 3.3.3 RANGE OF OBJECTIVES (2)
  - 3.3.4 WEIGHING OF CONSEQUENCES OF ALTERNATIVES (2)
  - 3.3.5 ASSESSMENT OF CONSEQUENCES OF ALTERNATIVES (3)
  - 3.3.6 RE-EXAMINATION OF DECISION-MAKING PROCESS (3)
  - 3.3.7 USER OF INFORMATION (3)
  - 3.3.8 CONSIDERATION OF IMPLEMENTATION AND CONTINGENCY PLANS (2)
  - 3.3.9 EFFECT ON GROUP DISCUSSIONS (3)
  - 3.3.10 EFFECT ON DECISION MAKERS' CONFIDENCE (2)

[NOTE: NUMBERS IN PARENTHESES ARE NUMBER OF QUESTIONS USED FOR ASSESSMENT OF THE DSS IN A SHORT-ANSWER QUESTIONNAIRE.]

and Donnell (1986) in order to evaluate the adequacy of five different decision support system prototypes, including three expert systems, developed to support U.S. Air Force tactical decision making. We use the term "subjective MOEs" because a questionnaire was used to assess the prototypes' performance on the MOEs. Consistent with research by Adelman (1982), Huber (1986), Shycon (1977) and others indicating that evaluators must monitor the compatibility of decision technology with the characteristics and needs of the organization, as well as the user, the hierarchy of MOEs is organized to measure the three interfaces represented pictorially in Figure 1-2.

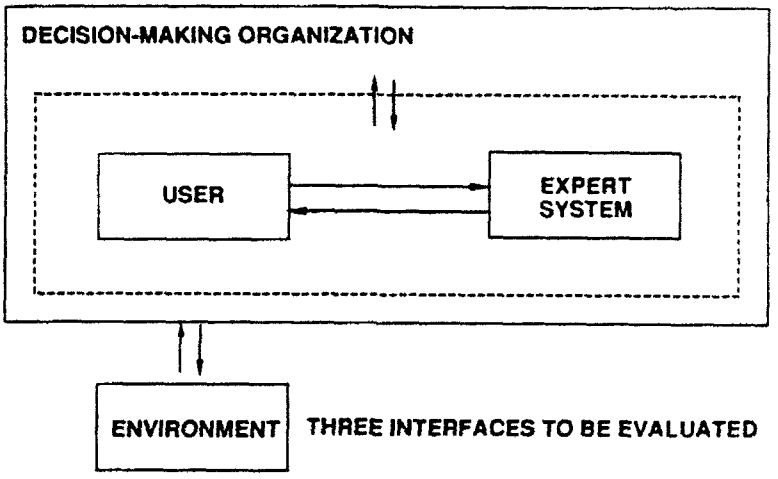


Figure 1-2: The Three Interfaces to Monitor and Evaluate when Developing Expert Systems

The first interface is between the expert system and the user (ES/U). Here the issue is the extent to which characteristics of the system facilitate or hinder its usability. The second interface is between the user (and expert system) and the larger decision-making organization (U/DMO) of which both are a part. Here the issue is to what extent the system facilitates the decision-making process of the organization. The third interface is between the decision-making organization and the environment (DMO/ENV). Here the issue is whether or not the expert system improves the quality of the organization's decision making and, in turn, the organization's overall performance.

As can be seen, the MOEs presented in Table 1-1 are organized into a hierarchy such that the three uppermost levels represent the three interfaces

in Figure 1-2. The topmost level of the hierarchy represents the expert system's overall utility or value to the decision maker and organization for whom it is being built. Each of the three uppermost levels of MOE categories is subdivided further until it is easy to identify distinctly measurable MOEs. By assuming that each terminal node in the hierarchy could be translated into an MOE, the task of evaluating an expert system is translated into one of "scoring and weighting." That is, one first tests the expert system on each of the bottom-level nodes of the evaluation hierarchy in order to obtain the system's scores on the MOEs. By then "weighting" these "scores" by the relative importance of the MOEs and MOE categories moving up the hierarchy, one obtains an explicit, retraceable process for evaluating the overall value, and relative strengths and weaknesses, of the expert system.

The MOEs in Table 1-1 will be considered in more detail in the case study. For now, it is important to make three points. First, the specific MOE(s) one selects for testing and subsequently evaluating one's expert systems should be determined from a decision-making perspective. What information is needed? Who needs it? What stage is the expert system development process in? In addition, one needs to consider how these questions, as well as potentially limiting factors (e.g., funds, time, personnel, etc.), affect the selection of testing methods. Remember, the selection of a particular method is a decision in and of itself, for methods differ on various dimensions (or attributes), such as the generalizability of their data to real-world settings, their costs, the amount of control the evaluator has in implementing them, etc. Testers and evaluators need to systematically consider the technical tradeoffs, limiting factors, and decision-making perspective when selecting test and evaluation methods.

Second, an eclectic approach is required to effectively test and evaluate expert systems. As Riedel and Pitz (1986) point out, many people erroneously assume that objective, empirical measurement is the most valid and, therefore, preferred type of data to collect. However, the preference for a particular type of data depends on the relative importance of the MOE being measured by that data. If the system's performance in solving test cases is the most important MOE, then objective empirical data will be the most important type of data to collect. However, if the user's opinion of the

expert system is the most important MOE, which it often is for systems designed to assist experts, then subjective data will be the most important type of data to collect. Moreover, as we pointed out earlier, aggregation of all the test data to make an overall evaluation of the expert system is inherently a subjective judgment.

It must be remembered that the expert system can be tested on many different kinds of MOEs. Different testing methods and, thus, types of data, are appropriate for different MOEs. For example, the methods used to test the logical consistency of the knowledge base are different from those used to test the user's performance with the expert system or how well the software is written or what the users think of the reasoning trace. The three-phased approach presented herein represents the kind of eclectic approach required to comprehensively test and evaluate expert systems. The goal of this book is to overview a range of subjective, technical, and empirical methods for testing an expert system on MOEs important to the sponsoring team.

Third, the hierarchy of MOEs presented in Table 1-1, when combined with relative importance weights, represents an application of Multiattribute Utility Assessment (MAUA). MAUA, as well as other subjective evaluation methods, will be considered in detail in this book. What is important to note here is that these methods can be used to evaluate the implications of the different tests from the sponsoring team's perspective. In doing so, it is the tester's job to test the expert system on each of the bottom-level MOEs in the hierarchy, and to indicate their relative importance from a technical perspective. However, it is the job of the sponsoring team (and users) to assign the relative importance weights to the MOE categories; for example, how important is the logical consistency of the knowledge base versus its functional completeness versus its predictive accuracy versus its integration with existing databases versus the user interface versus the system's response time versus the user's/organization's performance, etc. Both the tester and sponsoring team may be involved in assigning the degree of importance to different levels of performance within any given MOE category. The evaluator must work with members of the sponsoring team and users to make these "trade-off judgments" and, more generally, develop an explicit framework for relating

the multitude of specific tests to an overall evaluation of the system's value to the organization.

THE CASE STUDY

Over the twenty-four-month period from September, 1981, to September, 1983, PAR Technology Corporation was the prime contractor to the Rome Air Development Center (RADC) on a contract designed to develop five decision support system (DSS) prototypes for supporting U.S. Air Force (USAF) tactical decision making. Four tasks were performed on this project. Task I was a detailed study of the various activities, and their functions, performed in USAF tactical decision making. The study was performed with a view toward defining potential aiding situations in which the technologies of Artificial Intelligence, Decision Analysis, and Operations Research might be applied to aid decision making. In Task II, 28 prototypes were proposed for development. The proposals were subjected to a two-phase utility analysis and to a cost-benefit analysis in order to identify the five prototypes that would be developed on the project. These five DSS prototypes were developed by PAR and its subcontractors (Decisions and Designs, Inc. and Systems Control Technology, Inc.) in Task III, with different companies building different prototypes on the basis of the match between the technical requirements of the prototypes and the technical skills of company personnel. All five prototypes were evaluated in Task IV by a test and evaluation team led by the first author.

This section overviews the three-phased (i.e., technical, empirical, and subjective) evaluation of a DSS prototype developed by PAR called DART, which is an expert system (see Barth et al., 1983) to assist in activity node identification. The activity node identification process addressed by DART is extremely difficult to perform because of the varying nature of the nodes of interest and the tremendous volume of available relevant data. Because of limited time and potential information overload, experience has become an increasingly important factor in the activity node identification process. There are, however, few analysts with the necessary activity node identification experience. An expert system DSS prototype represented a means of capturing activity node identification expertise, and making it available to inexperienced analysts. The DART prototype was to contain enough expert

knowledge to identify (with a degree of certainty) thirteen different types of activity nodes. More importantly, the DART prototype had to be capable of effectively communicating the rationale for the identification, for it was to support the analyst's decision-making process, not replace it.

The results for each of the three evaluation phases are now considered, in turn. It is important to emphasize three general points here at the outset of the overview. First, there were limited funds and time to perform the tests and evaluations. All test and evaluation activities from initial planning, to conducting the tests and performing the analysis, and to documenting the results, had to be conducted for approximately 10% of the project's total cost. Moreover, the actual testing of all five prototypes had to be conducted within a seven-month period. Finally, the prototypes were tested sequentially, consistent with the participating contractors' development schedule. For example, DART was the second prototype developed and tested; it was tested in the second month of the testing period.

Second, consistent with the perspective of integrating test and evaluation results into development, each prototype was tested twice. The first test was with engineers at RADC who were novices in the prototype's domain area, but who had at least a college degree emphasizing computer science or engineering. The second test was with domain experts who represented potential users of the fully developed system. There were always at least two weeks between the two tests to provide the development team with some time to enhance the prototype based on the feedback obtained in the first test session.

And, third, the overall purpose of the evaluation of DART and the other four prototypes developed on the contract was to determine which ones showed the greatest potential value to the Air Force and, therefore, should go on to further development. Consequently, efforts were made to standardize the evaluations of the five prototypes as much as possible. This fact, plus time, money, and scheduling constraints for the evaluations, resulted in the decision to emphasize the subjective and empirical evaluation phases over the technical one.

In particular, each prototype was subjected to an experiment to test whether the aid significantly improved users' performance. Second, the table of subjective MOEs presented in Table 1-1 was used to obtain participants' opinions of the prototypes' strengths and weaknesses. Lastly, each test and evaluation session concluded with a round-table meeting between sponsoring team and the domain experts who evaluated the prototype in order to further help the sponsors assess whether the prototype was a good enough option to warrant further funding. This last point clearly illustrates that the test and evaluation team saw their overriding purpose to be providing the sponsors with the stimuli necessary to test the hypothesis that the prototype would (or would not) improve organizational effectiveness and, consistent with the SHOR paradigm, to select the appropriate option(s) for proceeding in the future.

The following overview is based on Adelman and Donnell (1986); more specific details can be found in Adelman and Gates (1983).

Technical Evaluation

The technical evaluation of the DART expert system prototype took place at PAR's corporate headquarters in New Hartford, New York, in late January, 1983. The first issue, which was actually considered early in the development process (Rockmore et al., 1982) was whether artificial intelligence was an appropriate analytical method to select for the activity node problem. The answer was an affirmative one. Consistent with the SHOR paradigm, the user's job is to evaluate and select hypotheses regarding activity nodes. Artificial intelligence is ideally suited for this requirement.

The technical evaluation focused primarily on the system characteristics of DART's many modules. These modules are represented in Figure 1-3 from a functional perspective. The most visible portion of the system is the Executive, which assists the user in managing the aid. The Executive consists of:

- The Inference Engine,
- The Advice Interpreter,

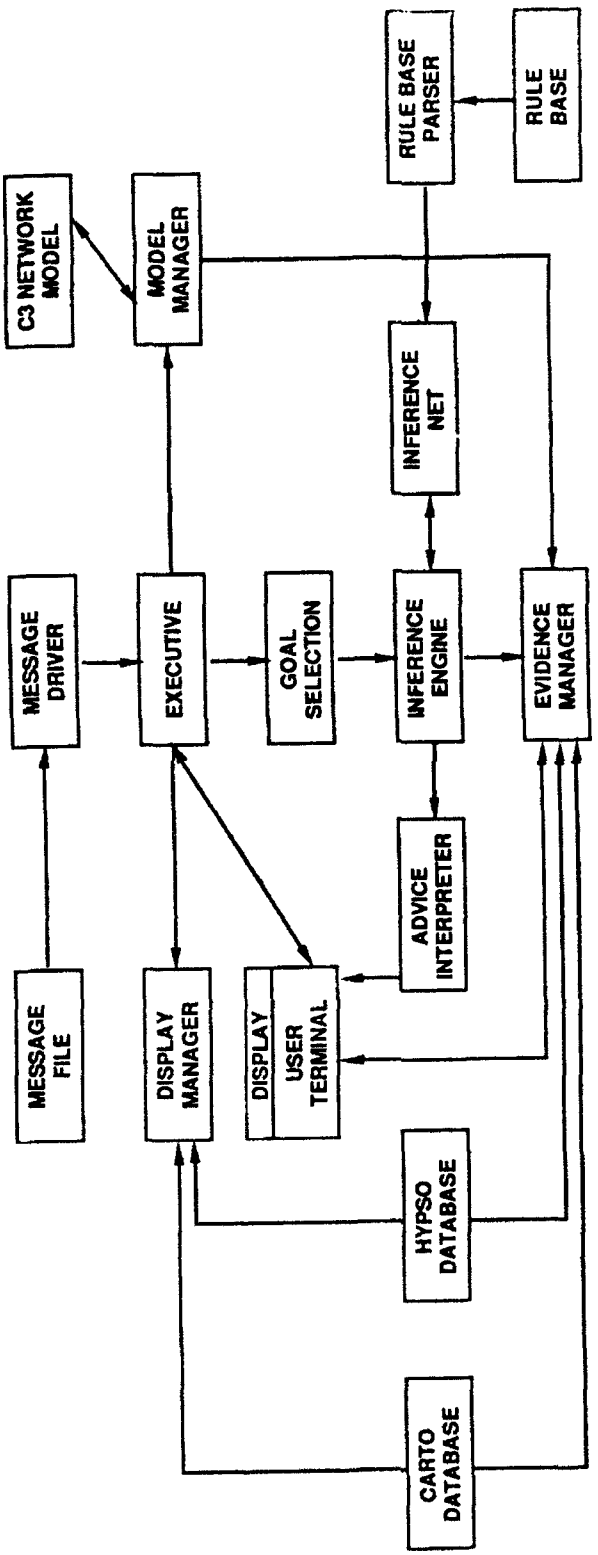


Figure 1-3: DART Functional Overview

- The Model Manager,
- The Display Manager.

Based upon a selected goal hypothesis (one of the thirteen identifiable activity nodes), the Inference Engine accesses that portion of the Inference Network which will analyze the pertinent, available information concerning the goal. The rules contained in this selected segment of the Inference Network use the data (or evidence) found in the message and associated degrees of belief from the Evidence Manager to identify the most likely activity node. The Advice Interpreter advises the user of the degree of belief for this identified activity node. Additionally, the user can consult the Advice Interpreter for the evidence used in reaching this decision. Once advised, the user can call the graphics display via the Display Manager or call the Model Manager to update the activity node identification model. The Display Manager provides the means to display terrain data; the Model Manager places identified activity nodes on this terrain. The Message File and Driver provide a time-sequenced list of reports which the analyst can use to correlate multiple reports of the same activity node, thereby increasing the confidence in the identified activity node.

In brief, the evaluation team concluded that, from a technical perspective, the DART prototype contained all of the modules necessary for a consultative expert system to support the activity node identification, decision-making process. The experts who participated in the empirical and subjective evaluations supported this position, for, although they recommended many improvements, they neither recommended additional modules nor deletions of those already developed for the DART prototype. The logical consistency, functional completeness, and predictive accuracy of the knowledge base were considered as part of the empirical evaluation with the domain experts.

Empirical Evaluation

The goal of the empirical evaluation phase was to objectively assess whether DART significantly improved the accuracy of analysts performing the target identification process. To accomplish this goal, an experiment was performed. The three independent variables were (1) whether the analyst was

experienced or not in activity node identification, (2) whether the analyst performed the activity node identification task with or without DART, and (3) which of two different activity node identification problems the analyst performed. The dependent variable was the quality of the analyst's solution to the activity node identification problem.

The test setting for the empirical evaluation was created concurrently with the performance of the technical evaluation. An isolated room 14 feet by 12 feet was used for the unaided condition. A smaller room with a computer terminal and DeAnza display, both of which were linked to a VAX 11/780 system, was used for the aided condition. [Note: Operational versions of DART and the other DSS prototypes were to be tailored for military microcomputers on subsequent procurements at the government's discretion.] Both test areas had 1:500,000 and 1:250,000 scale charts of the geographic area of interest used in the activity node identification problems.

The participant's task for each of the two problem scenarios was to identify ground components of opposing forces moving in a specified direction over the area of interest on the basis of message data. The problems differed in the number of each of thirteen possible activity node types and the available message data. In the first problem there were 100 messages; in the second problem there were 80 messages. Each participant had 1 1/2 hours to perform each problem regardless of whether he or she worked with or without DART. The activity nodes identified by each participant were placed on acetate and overlaid on the large wall map representing the geographic area for which the problem scenarios were created. Since a correct solution existed for each scenario, it was possible to determine the number, location, and type of correctly identified activity nodes. Using this information and looking at the acetate overlay map, the experts then rated the quality of each participant's solution for each scenario on a 0-to-10 scale, where higher scores meant a better solution. Qualitative ratings were required because all misclassifications were not equally detrimental; the solution's quality depended on an analyst's judgment as to the importance of the type and location of the misclassifications. Each participant's solution was coded by letter to minimize the experts' ability to identify its author.

The empirical and subjective evaluations were conducted at PAR's corporate headquarters in New Hartford, New York, over two 4-day periods in February and March, 1983. The participants for the first session were RADC personnel who had no activity node identification experience; these four participants are referred to as nonexperts. The participants for the second session were U.S. Air Force analysts with considerable activity node identification experience; these three participants are referred to as experts. The participants were provided through the cooperation and courtesy of different Air Force agencies. Although the sample size was small for an empirical evaluation, it was as large as could be obtained, given prior Air Force commitments. Larger sample sizes should be used whenever possible to provide the power necessary for traditional statistical tests of a prototype's effectiveness (e.g., see Adelman et al., 1982).

The primary value of the session with the nonexperts was identification of the following three necessary modifications to the test conditions and the DART user interface. First, the nonexperts did not have enough hands-on training in using DART; consequently, the experts' schedule was modified to provide more training. Second, DART was slow and cumbersome to use because it required the user to update the Model Manager and Display Manager after each message by sequentially accessing a number of menus; consequently, DART was modified to give the user the ability to automatically update the Model Manager and Display Manager after each message, thereby making DART much faster to use. And third, the message flow in the unaided participants' task was found to be unrepresentative of the analyst's actual environment; consequently, the message flow was modified for the session with the experts so that it better represented the analyst's actual environment.

The results of the session with the nonexperts were, however, not included in the empirical and subjective evaluations of DART because so many changes were made to the test conditions and DART user interface between sessions that, prior to the session with the experts, the test and evaluation team concluded that it was inappropriate to combine the results of the two sessions. However, it is of importance to note here that the cumulative effect of the three classes of problems described above resulted in the nonexperts performing the activity node classification task worse with than

without DART at a statistically significant ($p < .05$) level. *Integrating the feedback* from the tests with the nonexperts back into the development process will, as will be shown, *significantly improve the DART prototype*.

The schedule for the DART evaluation session with the experts proceeded as described below over the 4-day evaluation period. Monday morning was dedicated to providing a technical overview of DART so that the experts would understand how DART performed activity node identification. On Monday afternoon and most of Tuesday, the experts received hands-on training in using DART. This was accomplished by providing each expert with two 1 1/2 hour training sessions on DART. The DART test scenarios were completed by the experts on Tuesday and Wednesday. Two of the experts worked the first scenario in the unaided condition, and one used DART. In contrast, two experts worked the second scenario using DART and one worked without it. This arrangement ensured that each expert had used DART to solve one scenario, and that there were 3 aided and 3 unaided solutions in total. On Thursday, the experts rated the quality of the three solutions generated by the experts for each scenario. The experts' ratings were based on the number, location, and type of both correctly and incorrectly identified activity nodes. The participants also completed the evaluation questionnaires and discussed their impressions of DART's strengths and weaknesses with members of PAR's evaluation team and RADC personnel monitoring the contract.

The experts' quality ratings of the experts' solutions, and the conditions under which they were generated, are presented in Table 1-2. The higher the number, the better the quality rating. Pearson product-moment correlations (r) were calculated to determine the extent of agreement among the three experts' ratings. Pearson product-moment correlations can vary from +1.0 (indicating perfect agreement) to -1.0 (indicating perfect disagreement); a value of zero indicates that there is no relationship among the ratings. The Pearson product-moment correlations were computed by combining the ratings for both scenarios, thereby creating a sample size of six (instead of three) and, in turn, greater confidence in the results. The Pearson product-moment correlations among the quality ratings of experts E1 and E2, E1 and E3, and E2 and E3, were .94, .93, and .97 respectively. All three correlations were statistically significant at the $p < .01$ level, thereby indicating that there

was considerable agreement among the experts' quality ratings of the solutions.

Table 1-2: The Experts' Quality Ratings for their Solutions with and without DART

SCENARIO #1					SCENARIO #2				
GOB*	E1	E2	E3	Mean	GOB*	E1	E2	E3	Mean
A (Unaided)	3	5	3	3.67	A (Aided)	6	8	7	7.0
B (Unaided)	8	8	8	8.0	B (Unaided)	9	9	9	9.0
C (Aided)	5	7	7	6.33	C (Aided)	7	8	8	7.67

*GOB is Ground Order of Battle

The mean quality rating and the sample size for each of the four cells in the 2 (Aid) × 2 (Scenario) design for the experts' solutions are presented in Table 1-3. As can be seen, there are only three observations each in the Aided-Scenario I and Unaided-Scenario II cells. This occurred because, since only three experts participated in the evaluation, two cells of the design could have only one participant if each expert were to (1) perform each scenario only once and (2) work both with and without the aid. The Aided-Scenario I and Unaided-Scenario II conditions, and the expert who worked them, were randomly selected by the evaluation team. Table 1-3 shows a sample size of three observations for these two cells because each of the three experts independently evaluated the one expert's solution. The Unaided-Scenario I and Aided-Scenario II cells have a sample size of six observations because each of the three experts independently evaluated the two experts' solutions for these two cells. Each expert's rating was used as an independent observation of each solution, instead of taking the mean of the three experts' ratings for each solution, to have a sample size that even approached the size necessary for performing statistical tests.

Table 1-3: Experts' Mean Quality Ratings and Sample Size for the 2 (DSS) × 2 (Scenario) for the Experts' Solutions

	SCENARIO I	SCENARIO II	$\bar{x}$
AIDED	N = 3 6.33	N = 6 7.33	N = 9 7.00
UNAIDED	N = 6 5.83	N = 3 9.00	N = 9 6.89
$\bar{x}$	N = 9 6.00	N = 9 7.89	N = 18 6.95

A repeated measures t-test, where the experts were the repeated measure, was used to statistically determine whether, on the average, (1) experts performed better aided than unaided, and (2) if performance was significantly better for one scenario than the other. [*Note: An Analysis of Variance was* not used because, due to the small and unequal sample sizes for the cells, analysis of the Aid × Scenario interaction was not warranted.] There was no statistical difference in the mean scores for the aided and unaided conditions; experts performed equally well working with DART as without it. Mean performance was, however, significantly better for Scenario II than Scenario I ($t = 2.34$, $df = 4$, $p < 0.05$). This may have been due to practice effects because Scenario II was performed after Scenario I. This hypothesis is unlikely, however, for the participants were experts who, of course, had substantial experience performing substantially more complex scenarios in operational settings. A more likely explanation is that Scenario II was easier than Scenario I.

An additional analysis was performed in an effort to better understand why there was no difference in the performance of experts working with and without DART. This aspect of the empirical demonstration illustrates but one of a number of different methods that will be considered later in this book for testing the predictive accuracy of an expert system's knowledge base. In

particular, the evaluation team counted the number of mistakes the experts made for the thirteen different activity nodes in the two scenarios, both with and without DART. Although no statistical tests were performed because of the small size for each node, examination of the mean scores suggested that, when aided, the experts were better in identifying certain activity nodes, and worse in identifying others. This suggests that (1) DART's rule-base for identifying certain activity nodes needed improvement, and (2) that such improvement would result in experts performing the test scenarios better with DART than without it.

Subjective Evaluation

The subjective evaluation of DART was composed of the experts' answers to two questionnaires. The first questionnaire was of a short-answer format with the questions designed to assess the expert system's performance on the subjective MOEs presented in Table 1-1. The second questionnaire was of an open-ended format which gave the experts an opportunity to indicate, without any prompting from the evaluation team, what they perceived to be the strengths and weaknesses of the DART prototype and recommend improvements to it.

We will present only the results obtained from the first questionnaire for two reasons. First, there was general agreement between the answers to the two questionnaires; consequently, it is unnecessary to present the results to both of them here. Second, the short-answer questionnaire had been standardized so that, except for substantive changes unique to DART, the same questionnaire could be used to assess participants' impressions of the strengths and weaknesses of each of the five prototypes developed on the contract; consequently, the short-answer questionnaire represented the first step in developing an empirically-based questionnaire that could be used by other people evaluating decision support technology. This focus, as well as the detailed analysis of the questionnaire for all five of the prototypes developed on the contract, can be found in Adelman, Rook, and Lehner (1985).

Before describing the questionnaire, we will briefly describe the MOE hierarchy to facilitate readers' consideration of how to develop ones ap-

propriate to their development projects. In particular, the MOE categories were designed to be as general as possible so that the same MOEs could be used to evaluate each prototype. To accomplish this, Adelman and Donnell (1986) refined and expanded the hierarchy of evaluation criteria initially developed by Sage and White (1980) to be compatible with the three-interface perspective presented in the previous section of this chapter. In doing so, they used as many of the criteria as possible that were used earlier in the contract when deciding which prototypes to develop in the first place. Other MOEs could (and would), of course, be used in an evaluation, depending on the characteristics of the expert system and the concerns of members of the sponsoring and development teams.

MOEs assessing the quality of the Expert System/User interface were divided into two major groups of criteria: those that assessed the match between the expert system and potential user's background, workstyle, and operational needs; and those that assessed the adequacy of the expert system's characteristics. This latter group was composed of general expert system characteristics—such as its ease-of-use and response time—and specific characteristics—such as the adequacy of the expert system's knowledge base, graphic displays, hard-copy capabilities, and text.

MOEs assessing the quality of the User-Expert System/Decision Making Organization interface were divided into two major groups of criteria: those assessing the expert system's efficiency from an organizational perspective, and those assessing the system's fit into the organization. Efficiency criteria included the amount of time it took to use the expert system to accomplish the task it was supporting (this is distinctly different from its response time), data management and set-up time requirements, and, pertinent to the present application, the system's perceived reliability and supportability under battle conditions. Criteria explicitly focusing on the expert system's potential effect on organizational procedures, other people's work, the flow of information, and its value in performing other tasks were used to assess the system's fit into the organization for which it was being developed.

MOEs assessing the quality of the Decision Making Organization/Environment interface were grouped into three major criteria: the perceived quality of decisions obtained using the expert system; the extent to which the expert system's technical approach matched the technical requirements of the task; and the extent to which the system improved the quality of the decision-making process. This last group of criteria was quite broad, ranging from the extent to which the expert system helped the user survey a wide range of alternatives and objectives, to the degree to which the system increased or decreased the user's confidence in the decision.

We now turn to describing the short-answer questionnaire. Specifically, the questionnaire had 121 questions. Most of the questions assessed the bottom-level MOEs in Table 1-1; however, 6 questions directly assessed overall utility (node 0.0 in Table 1-1), 2 questions directly assessed decision process quality (node 3.3 in Table 1-1), and 3 questions each assessed the quality of the training sessions and the test scenarios (neither of which are MOEs). All questions required the participant to respond on a eleven-point scale from 0 (very strongly disagree) to 10 (very strongly agree), with 5 being "neither disagree nor agree."

There were two or more questions for each MOE criterion in an effort to achieve greater confidence in the criterion scores. The number in the parentheses to the right of each bottom-level MOE in Table 1-1 indicates the number of questions assessing that criterion. The actual number depended on the availability of previously written questions assessing the criterion (e.g., from Sage and White, 1980), the ease in writing "different-sounding" questions for the criterion, and its depth in the hierarchy.

Half the questions for each criterion were presented in each half of the questionnaire to eliminate sequence-ordering effects. In most cases, a high score indicated good performance, but, typically for one question measuring each criterion, a low score indicated good performance in an effort to ensure that the participants paid careful attention to the questions. A prototype's score on a bottom-level criterion was the mean score of the participants' responses to the questions assessing it. Values for criteria moving up the hierarchy were the mean score for the criteria below it.

It is important to make two technical notes at this point. First, by averaging lower-level criterion scores to obtain upper-level criterion scores, one is giving each MOE criterion equal weight in the hierarchy. Although it was quite possible that the participating domain experts may have thought that certain bottom-level criteria were more important than others, members of the evaluation team thought it inappropriate to have the (DART) experts differentially weight these criteria at the time of the evaluation because we wanted to use the same weights for evaluating all five prototypes in order to provide a common evaluation baseline. And, since the hierarchy of MOEs in Table 1-1 was substantially larger and in many ways different from the MOEs used in Task II of the project to select the prototypes for development, the evaluation team considered it inappropriate to obtain relative importance weights from the sponsoring team prior to (or during) testing and evaluation for fear that certain developers might consider their prototypes adversely affected. It is important to note here that subsequent research published by Adelman, Rook, and Lehner (1985) showed that, in general, the participating RADC engineers and USAF domain experts differentially weighted the importance of the criteria when assessing the overall utility of the prototypes.

Second, there is an alternative approach to obtaining the scores on the upper-level criteria. Specifically, one could have taken the average of the scores to all the questions assessing each upper-level criterion. For example, to obtain a score for criterion 1.2, one could have averaged the scores for all the questions assessing criteria 1.2.1 and 1.2.2 instead of just averaging the mean scores for criteria 1.2.1 and 1.2.2 as we did. The alternative approach would have given greater weight to criterion 1.2.2 because there were more questions for criterion 1.2.2 than for 1.2.1. Again, because we did not want to differentially weight the MOE criteria, we rejected this approach.

On the basis of the six questions directly asking about its utility, DART received a mean score of 8.22 on the 0-to-10-point scale. On the basis of the evaluation hierarchy, DART received a mean overall utility score (node 0.0) of 7.36. The Expert System/User interface received the highest mean score (7.81) of the three interfaces. The User-Expert System/Organization

interface (7.17) and Organization/Environment interface (7.09) received comparable scores.

The experts' subjective evaluation scores for all of the criteria in the MOE hierarchy are presented in Table 1-4. From the perspective of the quality of DART's knowledge base, it is important to note that, relatively speaking, the experts gave low scores to the expert judgments stored in the system (node 1.2.2.3 = 6.84), DART's technical soundness (node 3.2 = 6.52), and the framework for incorporating judgments (node 3.3.1 = 6.83). Although these scores are still good, they are consistent with the results of prototype's empirical evaluation, which was that DART's knowledge base still needed work.

CHAPTER SUMMARY

The purpose of this chapter was to present a general framework and approach for testing and evaluating expert systems, and to show how they were applied in a single case study. The framework provides a paradigm for considering decision making and, hence, expert systems, within the broader organizational, problem-solving context within which both exist. The approach is multi-faceted in that it has three phases: a technical phase for "looking inside the black box;" an empirical phase for rigorously assessing the expert system's impact on performance; and a subjective phase for obtaining users' opinions regarding the system's strengths and weaknesses. Finally, the case study presented the procedures for, and results of, implementing the general framework and three-phase approach for an Air Force expert system prototype to support opposing force activity node identification.

This chapter has been introductory in nature, for we have tried to emphasize a general perspective that one should keep in mind when testing and evaluating expert systems. Remember, test and evaluation methods are tools that can be used to provide sponsoring and development team members with the feedback they need to improve the judgments and decisions inherent in system development. For that reason, test and evaluation represents a control mechanism that keeps a requirements-driven, expert system development process on track.

Table 1-4: Experts' Subjective Evaluation Scores for DART on each MOE in the MAUA Hierarchy

0.0	UTILITY (I.E., POTENTIAL FOR IMPLEMENTATION)	
	- BASED ON 6 QUESTIONS	8.22
	- BASED ON THE MAUA CRITERIA HIERARCHY	7.36
1.0	DSS/USER INTERFACE	7.81
1.1	MATCH BETWEEN DSS & PERSONNEL	7.93
1.1.1	MATCH WITH TRAINING & TECHNICAL BACKGROUND	8.44
1.1.2	MATCH WITH WORKSTYLE/ WORKLOAD/INTEREST	7.08
1.1.3	MATCH WITH OPERATIONAL NEEDS	8.27
1.2	DSS' CHARACTERISTICS	7.68
1.2.1	GENERAL CHARACTERISTICS	8.46
1.2.1.1	EASE OF USE	8.56
1.2.1.2	TRANSPARENCY (I.E. USER UNDERSTANDING)	7.78
1.2.1.3	EASE OF TRAINING	8.83
1.2.1.4	RESPONSE TIME	8.67
1.2.2	SPECIFIC CHARACTERISTICS	6.90
1.2.2.1	USER INTERFACE	7.67
1.2.2.2	TYPES OF DATA FILES	6.56
1.2.2.3	EXPERT JUDGMENT STORED IN DSS	6.84
1.2.2.4	ABILITY TO MODIFY JUDGMENTS	6.88
1.2.2.5	DSS' AUTOMATIC CALCULATIONS	7.16
1.2.2.6	DSS' GRAPHS	5.84
1.2.2.7	THE NEED FOR HARD COPY	(9.33)
1.2.2.8	DSS' TEXT	7.33
2.0	USER-DSS/ORGANIZATION INTERFACE	7.17
2.1	EFFICIENCY FACTORS	6.65
2.1.1	SPEED	6.54
2.1.1.1	TIME REQUIRED FOR TASK ACCOMPLISHMENT	7.11
2.1.1.2	TIME REQUIRED FOR DATA MANAGEMENT	6.20
2.1.1.3	SET-UP TIME REQUIREMENTS	6.33
2.1.2	PERCEIVED RELIABILITY UNDER BATTLE CONDITIONS	6.50
2.1.3	PERCEIVED SUPPORTABILITY UNDER BATTLE CONDITIONS	6.91
2.1.3.1	SKILL AVAILABILITY	7.56
2.1.3.2	HARDWARE AVAILABILITY	6.25
2.2	MATCH BETWEEN DSS & ORGANIZATION	7.69
2.2.1	EFFECT ON ORGANIZATION PROCEDURES	7.00
2.2.2	EFFECT ON OTHER PEOPLE'S POSITION	7.42
2.2.2.1	POLITICAL ACCEPTABILITY	7.67
2.2.2.2	OTHER PEOPLE'S WORKLOAD	7.17
2.2.3	EFFECT ON INFORMATION FLOW	8.17
2.2.4	SIDE EFFECTS	8.17
2.2.4.1	VALUE IN PERFORMING OTHER TASKS	7.50
2.2.4.2	VALUE TO SAC OR OTHER SERVICES	8.33
2.2.4.3	TRAINING VALUE	8.67
3.0	ORGANIZATION/ENVIRONMENT INTERFACE	7.09
3.1	DECISION QUALITY	7.58
3.2	TECHNICAL SOUNDNESS (MATCH BETWEEN DSS' TECHNICAL APPROACH & ANALYSTS' TECHNICAL REQUIREMENTS)	6.52
3.3	DECISION PROCESS QUALITY (BASED ON 10 ATTRIBUTES)	7.18
	- BASED ON 2 QUESTIONS	(7.83)
3.3.1	FRAMEWORK INCORPORATING JUDGMENT	6.83
3.3.2	SURVEY RANGE OF ALTERNATIVES	7.33
3.3.3	SURVEY RANGE OF OBJECTIVES	6.83
3.3.4	WEIGHING CONSEQUENCES	7.33
3.3.5	ASSESSMENT OF CONSEQUENCES	6.67
3.3.6	RE-EXAMINATION OF DECISION-MAKING PROCESS	7.11
3.3.7	USE OF INFORMATION	7.78
3.3.8	IMPLEMENTATION	7.50
3.3.9	EFFECT ON GROUP DISCUSSION	7.11
3.3.10	CONFIDENCE	7.33

It must be remembered that the expert system can be tested on many different kinds of MOEs. Different testing methods and, thus, types of data, are appropriate for different MOEs. The specific MOEs one selects for testing and subsequently evaluating expert systems, should be determined from a decision-making perspective. What information is needed? Who needs it? What stage is the expert system development process in? In addition, one needs to consider how these questions, as well as potentially limiting factors (e.g., funds, time, personnel, etc.) affect the selection of testing methods. Remember, the selection of particular test and evaluation methods is a decision in and of itself.

The next chapter overviews subjective, technical, and empirical test and evaluation methods. Prior to doing so, however, we overview the expert system development process in order to better indicate the appropriateness of different test and evaluation methods during development. After overviewing the different kinds of methods, we present a (1) hierarchy of MOEs for capturing subjective, technical, and empirical data, and (2) general evaluation approach for integrating the tests on these MOEs into an overall assessment of the utility of the expert system under development.

CHAPTER 2:

LAYING THE FOUNDATION: TEST AND EVALUATION CRITERIA, THE EXPERT SYSTEM DEVELOPMENT CYCLE, AND AN OVERVIEW OF SUBJECTIVE, TECHNICAL, AND EMPIRICAL TEST AND EVALUATION METHODS

This chapter builds the foundation upon which the remaining chapters of this book rest. It has three principal sections. The first section overviews test and evaluation criteria identified by Adelman and Ulvila (in press). These criteria are organized into a hierarchy that can be used with Multi-Attribute Utility Assessment (MAUA) procedures—one of the subjective test and evaluation methods—to assess how well an expert system is meeting the requirements of users and sponsors. As was emphasized in the last chapter, the specific criteria one would use in one's tests and evaluations would depend on the specific requirements of one's users and sponsors. However, the hierarchy presented herein contains the wide range of test and evaluation criteria commonly found in the literature and, therefore, can give one a broad list of criteria from which to start. In Chapter 7, we provide some guidance for sifting through this list.

The second section of the chapter overviews the expert system development approach. Surprisingly, this "approach" takes somewhat different forms depending upon who is describing it. Nevertheless, it is epitomized by an iterative, prototyping approach that is distinctly different from the traditional software development process, although more recent formulations attempt to integrate requirements analysis and structured design aspects of the latter. Moreover, test and evaluation are an inherent part of this iterative development cycle. To quote Harrison (1989, p. 311), "Note that incremental development, refinement, reintegration and so on all imply that evaluation is continuous and inseparable from development."

Evaluation is continuous and inseparable from development because judgment and decision making are inherent parts of the process. Formal test and evaluation methods have to be capable of improving these judgments and decisions and, thereby, the development process, throughout its iterative life cycle. This requires the application of different test and evaluation methods, appropriately matched to the information and decision needs of

different members of the sponsoring and development teams, throughout development.

The third section overviews the many different subjective, technical, and empirical test and evaluation methods. Other chapters in the book will describe these methods in more detail. Our purpose here is simply to introduce the reader to these methods. Try to keep the broad evaluation perspective provided by the SHOR paradigm in mind when overviewing these methods. Remember, at the broadest level, the evaluator's job is to help members of the sponsoring team decide whether development of an expert system is an effective option for dealing with hypotheses regarding the current and/or future problem environment with which the organization will be dealing and, if so, the general requirements that the expert system will have to satisfy.

Once the development process is underway, the evaluator's job is to systematically gather, filter, and aggregate data (i.e., stimuli) about the expert system in order to test the hypothesis. If there is a problem or if it is not clear what action to take, then options need to be generated, evaluated, and selected for correcting the situation so that the development process can be kept on track. In short, the application of formal test and evaluation methods helps members of the sponsoring and development teams monitor the perceived utility of the expert system under development and take corrective action to increase the probability of its use and effectiveness.

All three classes of test and evaluation methods are applicable during a formal test and evaluation of an expert system prototype by an outside group, as was shown in Chapter 1. In addition, however, specific methods are more or less applicable at other times in the development cycle. In particular, subjective evaluation methods are applicable early in the cycle because they represent an explicit means for defining the judgments of members of the sponsoring team and potential users of the system. For example, Rockmore et al. (1982) used MAUA, and a MAUA-based cost-benefit analysis, to select among various types of DSS technology, including expert systems, for subsequent development. Slagle and Wick (1988) used a subjective method analogous to MAUA to evaluate candidate expert system application domains. And Bahill et

al. (1988) used MAUA to address the valuative and technical judgments inherent in selecting expert system shells.

Technical test and evaluation methods are also applicable during design and development. For example, as part of the knowledge elicitation and representation process, one should routinely assess the adequacy and accuracy of the knowledge base. This can be accomplished by using (1) static testing to help assess the knowledge base's logical consistency and completeness, and (2) experts, both those participating in development and those acting as evaluators, to help assess the knowledge base's functional completeness and predictive accuracy. Ideally, the test and evaluation team will also have access to "ground truth" data for assessing predictive accuracy, as illustrated in the DART evaluation. In addition, traditional software test and verification methods can be used to help assess the "service" versus "competency" requirements of the expert system. These methods have considerable applicability (a) prior to programming code for verifying requirements analysis documentation and functional models of the software), and (b) once the development process is well underway, during system packaging and transfer.

In contrast to technical test and evaluation procedures, which focus on how well the system was developed, empirical test and evaluation methods focus on how well decision makers can perform their task(s) with (versus without) the system. From an iterative, prototyping perspective, it is anticipated that experiments will be conducted throughout development as a means of objectively measuring the performance of the expert system and testing hypotheses for improving it. After transferring the expert system to the test organization, experiments, quasi-experiments, and case studies can be used to evaluate performance in the actual or simulated organizational setting. Remember, the expert system may be addressing only part of a much larger organizational decision. Even if the technical evaluation of the knowledge base shows that it has perfect predictive accuracy, the expert system's contribution still may not ensure better decision making within the larger organizational setting.

TEST AND EVALUATION CRITERIA

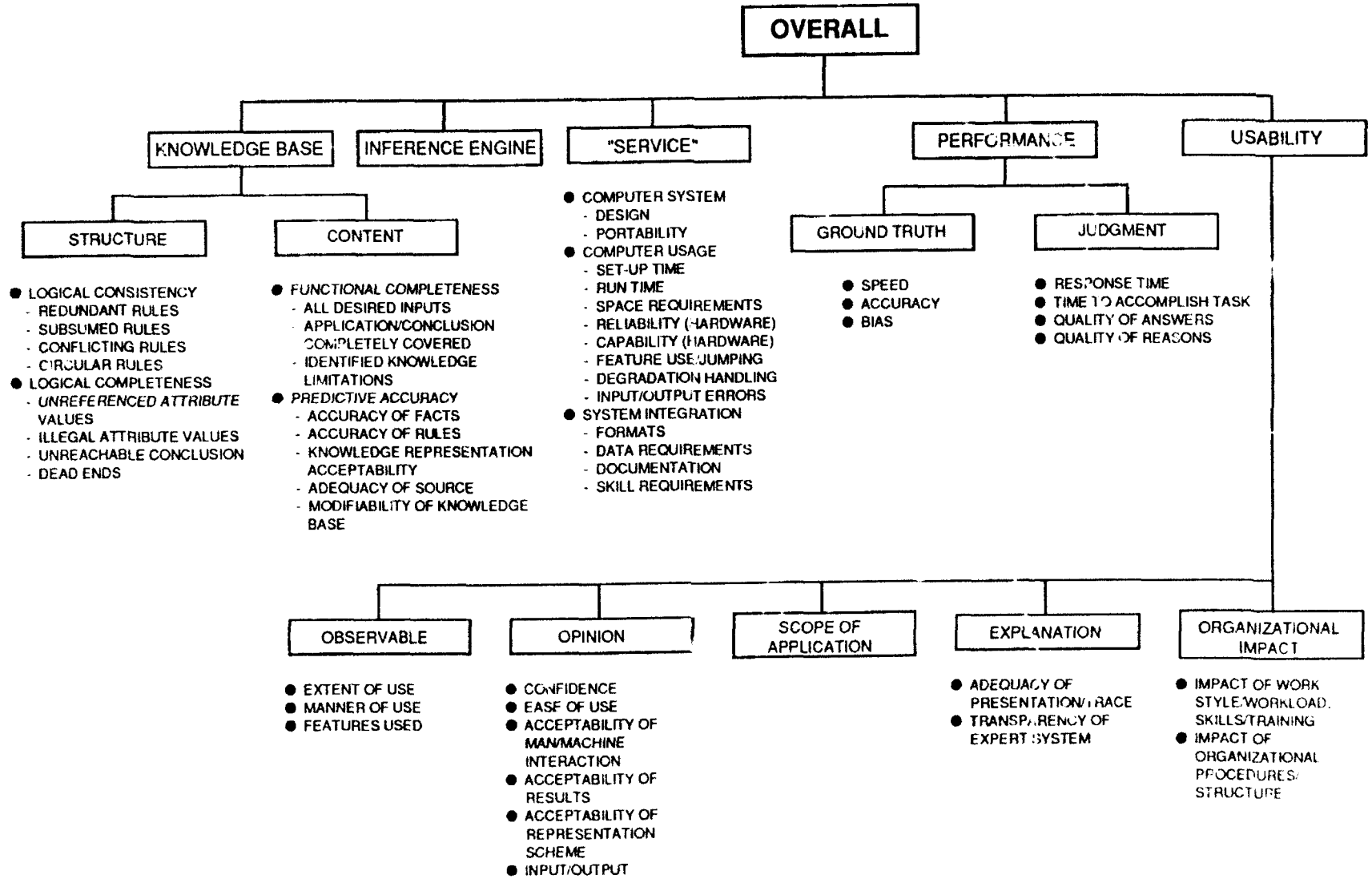
Table 2-1 presents a hierarchy of test and evaluation criteria. As will be discussed later in Chapter 3, this hierarchy can be used in conjunction with MAUA scoring and weighting procedures to evaluate the overall utility of an expert system to users and sponsors. Since the goal of developers and sponsors of expert systems (or for that matter, any type of software) is the creation of high-utility technology, the ultimate goal of test and evaluation is to determine (and facilitate) the extent to which this goal has been achieved.

The hierarchy in Table 2-1 has five branches. The first presents criteria for assessing the adequacy of the knowledge base or, as Rushby (1988, p. 75) has called them, the "competency requirements" of an expert system. Specifically, Table 2-1 lists different criteria for assessing logical consistency and completeness, functional completeness, and accuracy (and adequacy). Chapter 4 will focus on technical evaluation methods for measuring these criteria. In particular, we will overview the use of static testing methods for assessing the logical consistency and completeness of the knowledge base, and the use of domain experts, in conjunction with empirical evaluation concepts and methods, for assessing the functional completeness and predictive accuracy of the knowledge base.

The second branch is the correctness of the inference engine. Sponsors and users need to know that the inference engine has no errors in how it accesses the knowledge base and in how it propagates rules and probabilities (or other quantitative representations of uncertainties) in reaching conclusions. Testers should not assume that the inference engine has no errors. Some expert system shells do not provide test data with their documentation.

The third branch addresses conventional software requirements, referred to as "service requirements" by Rushby (1988, p. 75) within the context of expert systems. Conventional software test and verification criteria are important for expert systems too, particularly if the expert system has to be embedded in, or interfaced with, more conventional software modules. Service requirements include information about computer system design and portability

Table 2-1: A MAU Framework for Integrating Test and Evaluation Criteria



(i.e., transferability to other hardware and software environments), computer usage (e.g., set-up time, run time, space requirements, etc.), system integration, operator skill requirements, and documentation.

The fourth branch in Table 2-1 contains performance criteria, which are decomposed into criteria based on ground truth (or experts' ratings of decision quality), and the judgments of users. Both "ground truth" and "judgment" categories focus on the quality and speed aspects of performance. As Cats-Baril and Huber (1987) have shown, users' judgments of system performance do not always agree with more objective data. Consequently, although it is substantially more work, the use of ground truth data is urged when assessing performance. Moreover, we recommend here (as we will throughout the discussion of empirical evaluation methods) that experiments with aided and unaided corditions be relied on prior to transferring the system to its operational environment in order to rigorously assess performance with and without the system. Field experiments, quasi-experiments, and case studies should be relied on after transferring the expert system to its operational (or operational test) environment.

The usability branch is composed of criteria based on evaluators' observation of participants working with the system and participants' judgments of it. In conjunction with observation methods, users' key strokes can be recorded in an effort to better understand the extent to which the participants actually use the expert system during the problem-solving task, the manner in which they use it within the context of the more familiar procedures typically found in their job setting, and the specific features of the system they use most frequently. In addition, questionnaires analogous to the one used in the DART test and evaluation can be used to obtain users' opinions regarding their confidence in the expert system's recommendations, its ease of use, the acceptability of the person/machine interaction process, its scope of application, the adequacy of the system's explanations for its recommendations, the system's organizational impact, and specific input-output considerations.

The above criteria can be used to assess the adequacy of an expert system from the users' (and sponsors') perspective. To do so, one must test and, in

turn, score the expert system on each criterion that is considered important (i.e., given a nonzero, relative weight) by the users. Thus, each criterion represents a reference point that can be used to assess the system's progress on that criterion throughout development. For example, one would like to see a smaller number of redundant rules, a higher percentage of accurate predictions, better overall performance by the user, more favorable opinions about the interface, etc., as the system matures. Graphs can be developed to track trends on each criterion over time and, thereby, facilitate management of the test and evaluation process.

As we mentioned in Chapter 1, we will use the term "test" to refer to the process of measuring the expert system's performance against evaluation criteria, such as those listed in Table 2-1. The type of testing method to be used will depend on the criterion against which the expert system is being tested. For example, one would use (1) methods implementing the rules of logic to test the logical consistency of the rules in the knowledge base; (2) empirical data collection methods to test the predictive accuracy of the knowledge base against the judgmental accuracy of experts or ground-truth measures of accuracy; (3) observation methods to record the features of the expert system that users routinely use when solving test cases; and (4) subjective methods, such as questionnaires, to assess users' opinions of the system's strengths and weakness. That is why a multi-faceted approach is necessary to comprehensively test and evaluate expert systems.

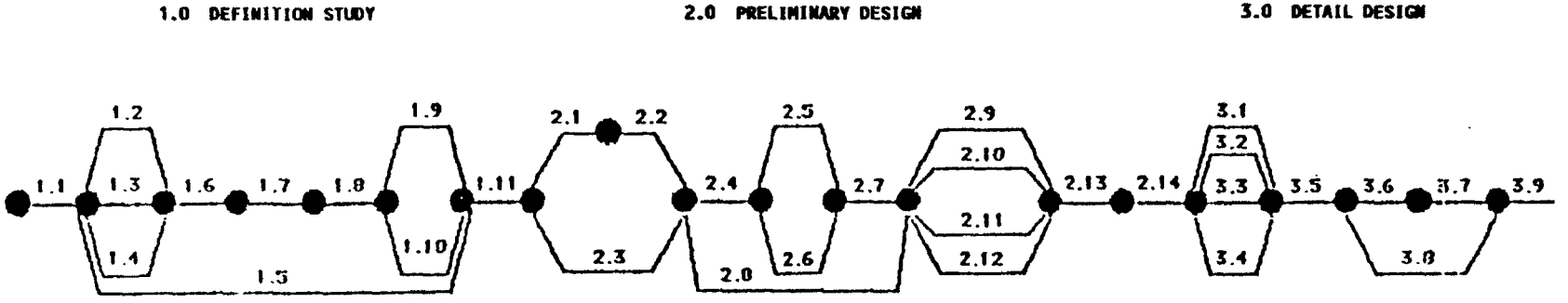
Central to the concept of evaluation is the concept of the utility or importance of the test results to the sponsoring team. For example, if the sponsoring team's primary requirements were that the expert system's (a) knowledge base be logically consistent, (b) functionally complete for the domain of interest, and (c) highly accurate in its predictions, then they probably could care less if its user interface was easy to use. As long as the expert system tested high on its primary requirements, then it would be evaluated as performing well. As this example illustrates, the process of aggregating all the different tests in order to reach an overall conclusion about the expert system is inherent in the term "evaluation." One of the very nice things about MAUA as a test and evaluation method is that it provides explicit, defensible procedures for converting the test results on many

different criteria into one common scale; that is, it provides a method for converting all the "apples and oranges" into a single overall evaluation.

We will briefly overview in Chapter 3 how MAUA procedures can be used to evaluate an expert system on the criteria shown in Table 2-1. We want to close this first section, however, by again emphasizing that evaluation is inherently a subjective process. Relative importance weights represent personal judgments. There is no mechanical procedure that can replace this fact or the use of other decision rules. Moreover, these judgments are appropriately the province of the decision makers, or their representatives, who are sponsoring the development of the expert system, and not the testers. For it is the sponsors of the development effort that have to make the final decision (i.e., evaluation) as to whether or not the expert system will fulfill their needs. Such a perspective is, of course, quite consistent with the SHOR paradigm.

THE EXPERT SYSTEM DEVELOPMENT APPROACH

Perhaps the best way to begin a description of the expert system development approach is to first present its counterpart, the "conventional" system design approach. As Andriole (1989) points out, there are various representations of the conventional approach. The one by Hice et al. (1978), shown in Figure 2-1, is an excellent example because it emphasizes the comprehensive, structured, and sequential nature of the conventional approach. Its strength is its procedural comprehensiveness. Its weakness, however, is its failure to extensively involve the user throughout the design process, and its relative inflexibility, which is best illustrated by the lack of feedback loops between the seven steps (and particularly the first two) in Figure 2-1. It is important to note that this presumed weakness is not always a weakness; it depends on the nature of the problem and tasks for which the system is being developed. The conventional approach can be very effectively applied when the problem and tasks are easily identified, defined, and structured. It is only when such clarity is elusive that other approaches need to be considered by developers.

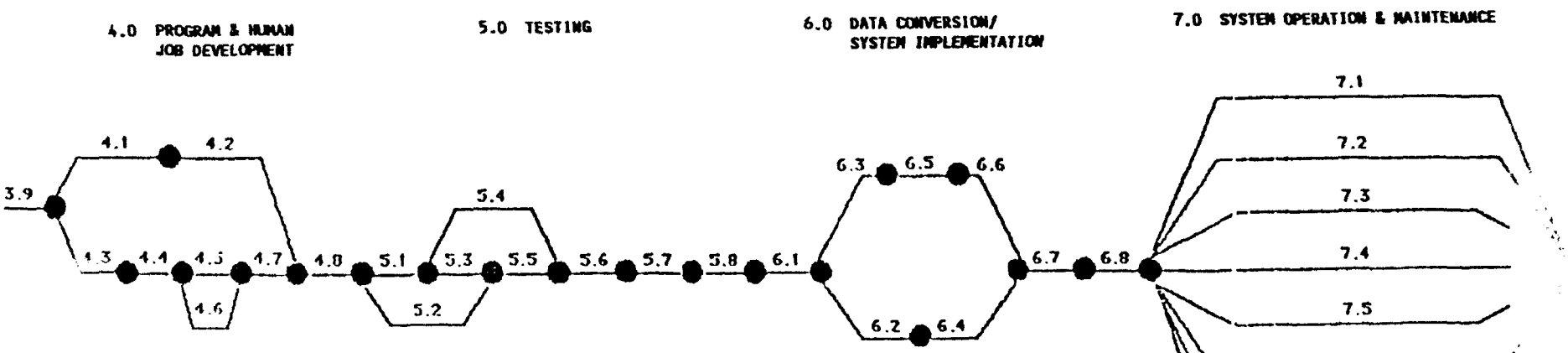


1.1 DEFINE THE PROBLEM/ESTABLISH STUDY SCOPE
1.2 COLLECT DATA ON EXISTING METHODS/ PROCEDURES
1.3 ANALYZE EXISTING METHODS/PROCEDURES
1.4 DEVELOP SYSTEM OBJECTIVES/SET PERFORMANCE CRITERIA
1.5 DETERMINE RESOURCES/CONSTRAINTS/ ASSUMPTIONS/ITEMS FOR RESOLUTION
1.6 SPECIFY SYSTEM OUTPUTS/INPUTS/ FUNCTIONS
1.7 DETERMINE SYSTEM CAPABILITY REQUIREMENTS/POTENTIAL APPROACHES
1.8 EVALUATE/SELECT SYSTEM APPROACH
1.9 DETERMINE IMPLEMENTATION/CONVERSION/ ACCEPTANCE REQUIREMENTS
1.10 PREPARE OVERALL PLAN/COST-BENEFITS ANALYSIS OF PROPOSED SYSTEM
1.11 PRODUCE DEFINITION STUDY REPORT

2.1 SPECIFY SYSTEM EXPANSION REQUIREMENTS
2.2 DEFINE OVERALL SYSTEM ENVIRONMENT
2.3 DESCRIBE SUBSYSTEMS
2.4 DEVELOP SUBSYSTEM INPUT/OUTPUT/ INTERFACE REQUIREMENTS
2.5 PREPARE SYSTEM/SUBSYSTEM FLOWCHARTS
2.6 DEVELOP PROCESS DESCRIPTIONS
2.7 SPECIFY SYSTEM PROTECTION REQUIREMENTS
2.8 IDENTIFY HUMAN ENGINEERING PROBLEM AREAS
2.9 DESIGN LOGICAL DATABASE STRUCTURE/ DEFINE ACCESS TECHNIQUES
2.10 SPECIFY DATA COMMUNICATIONS REQUIREMENTS
2.11 SPECIFY HARDWARE CONFIGURATION
2.12 SPECIFY SYSTEM SOFTWARE
2.13 PREPARE DEVELOPMENT/IMPLEMENTATION PLAN
2.14 PRODUCE PRELIMINARY DESIGN REPORT

3.1 DEVELOP HUMAN PROCEDURES
3.2 DESIGN MANUAL FORMS & COMPUTER INPUT/ OUTPUT INTERFACES
3.3 DESIGN PHYSICAL DATABASE
3.4 DESIGN SUBSYSTEM PROTECTION FEATURES
3.5 DEFINE SUBSYSTEM PROGRAMS
3.6 DEVELOP LOGIC FLOWCHARTS/TABLES
3.7 SPECIFY SOFTWARE UTILITIES/COMMON ROUTINES
3.8 DEVELOP SUBSYSTEM TEST PLAN
3.9 PRODUCE DETAIL DESIGN REPORT

Figure 2-1: Hice et al.'s System Development Methodology (Steps 1 through 3)



4.1 SYNTHESIZE POSITION DESCRIPTIONS FOR PERSONNEL SUBSYSTEMS
4.2 ESTABLISH PERSONNEL/ENVIRONMENTAL REQUIREMENTS
4.3 DEVELOP DETAILED PROGRAM FLOWCHARTS
4.4 CODE PROGRAMS
4.5 PREPARE SOURCE DECKS & COMPILE/ ASSEMBLE PROGRAMS
4.6 PREPARE PROGRAM (MODULE) DEBUG DATA
4.7 DEBUG PROGRAMS
4.8 PREPARE PROGRAM/HUMAN JOB DEVELOPMENT REPORT

5.1 DEVELOP DETAILED TEST PLAN/PROCEDURES
5.2 PREPARE SITE & INSTALL HARDWARE/ FACILITIES
5.3 DETERMINE RUN TIME ENVIRONMENT
5.4 TEST TRAINING COURSES/WORK AIDS/HUMAN PROCEDURES
5.5 BUILD TEST DATABASE/TRANSACTION FILES
5.6 TEST SUBSYSTEM/SYSTEM
5.7 PERFORM ACCEPTANCE TEST
5.8 PRODUCE TEST RESULTS REPORT

6.1 ESTABLISH CONVERSION/IMPLEMENTATION CONTROL PLAN/SCHEDULE
6.2 TRAIN OPERATIONS PERSONNEL ON HARDWARE/SOFTWARE/NEW SYSTEM
6.3 COMPLETE GUIDES FOR NEW SYSTEM
6.4 PERFORM DATA CONVERSIONS
6.5 CONDUCT MANAGEMENT ORIENTATION ON NEW SYSTEM
6.6 CONDUCT USER PERSONNEL ASSIGNMENT/ TRAINING FOR NEW SYSTEM
6.7 TRAIN MAINTENANCE TEAM ON HARDWARE/ SOFTWARE/NEW SYSTEM
6.8 TURN OVER SYSTEM & DOCUMENT CONVERSION/IMPLEMENTATION

7.1 DEVELOP/MONITOR CRITICAL INDICATORS
7.2 SCHEDULE MAINTENANCE ANALYSIS/PROGRAMMING TASKS
7.3 SCHEDULE COMPUTER OPERATIONS
7.4 PREVENT/RECOVER FROM RUN FAILURES
7.5 MONITOR DISASTER CONTROL/SECURITY PLANS
7.6 PROCESS CHANGE REQUESTS/DISSEMINATE CHANGE DOCUMENTATION
7.7 PERFORM ADDITIONAL TRAINING
7.8 REVIEW STATUS OF SYSTEM & ESTABLISH ANNUAL PLANS FOR OPERATIONS/MAINTENANCE

Figure 2-1: Hice et al.'s System Development Methodology (Continued)
(Steps 4 through 7)

Clarity of problem and task definition and, most importantly, structure, is typically elusive when starting an expert system development effort. The reason is that expert knowledge and reasoning represents the essential task structure of an expert system, and this is difficult to define and understand at the start of the development effort. To quote Harmon et al. (1988), "It's not that experts will not explain what they do, it's that they can't. Knowledge engineers must work patiently through a discovery process with human experts to develop and then enhance the system. No neat phases result in products that will not be reconsidered in subsequent phases. The original rules the knowledge engineers develop may later be rewritten entirely or dropped, as the experts and knowledge engineers gradually refine their understanding of the knowledge that must go into the knowledge base." To quote Cholawsky (1988, p. 42), "The conventional wisdom about this process is that expert systems development is necessarily an experimental process." It is one that emphasizes iteration, test and evaluation, and subsequent refinement.

For the above reasons, expert system developers and theoreticians have emphasized the application of prototyping methods with corresponding changes in the development process. The purpose of prototyping is to quickly develop a working model of the expert system and get the expert's and user's reaction to it in order to find out if the development process is on track. To quote Cholawsky (1988, p. 42), "Application is quickly followed by some initial prototyping effort. The prototype often serves as a combined feasibility study, design document, and functional specification effort. As the problem area becomes better understood, more involved prototype efforts are undertaken with more complex implementation, testing, and evaluation. The development team iteratively enhances each prototype until an operational system evolves. In the final development phase, the team is challenged to maintain and enhance end results of the prototyping efforts, transforming the mature prototype into an operational system."

Figure 2-2 presents Cholawsky's (1988, p. 44) representation of the "traditional expert system development methodology."

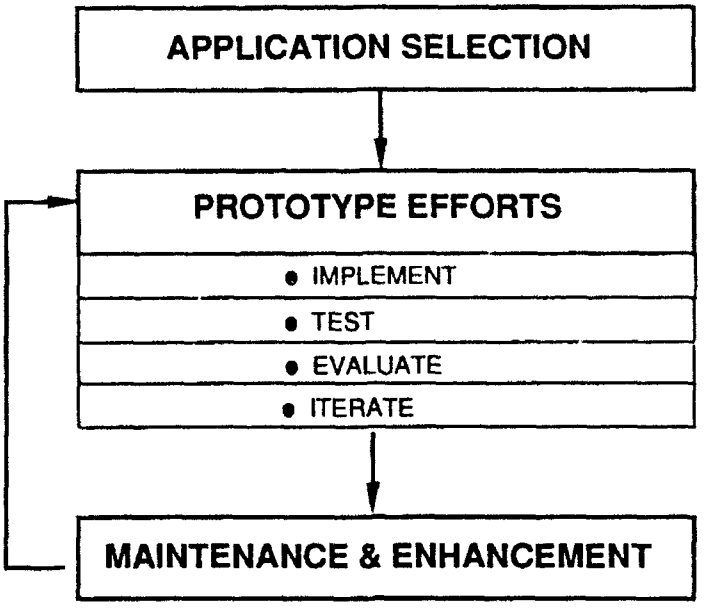


Figure 2-2: Cholawsky's (1988) Representation of the Traditional Expert System Development Methodology

Figure 2-3 presents Harmon et al.'s (1988) representation of the approach. As can be seen, Harmon et al. emphasize the constant interaction with the experts and users that is inherent in the prototyping approach. Constant interaction with users during prototyping is just as important as constant interaction with experts for two reasons. First, "users don't know what they want or need, but they do know what they like." And, second, "it is a lot easier to answer the question 'How do you like X?' than to answer the question 'How would you like X?'" (Hurst et al., 1983, p. 128). These reasons hold even when the participating expert is the designated user of the expert system. Consequently, in contrast to the conventional development approach, prototyping greatly expands the users' involvement in the development process by putting them in the explicit role of evaluating actual working representations of the expert system, and indicating how they should be modified, throughout development.

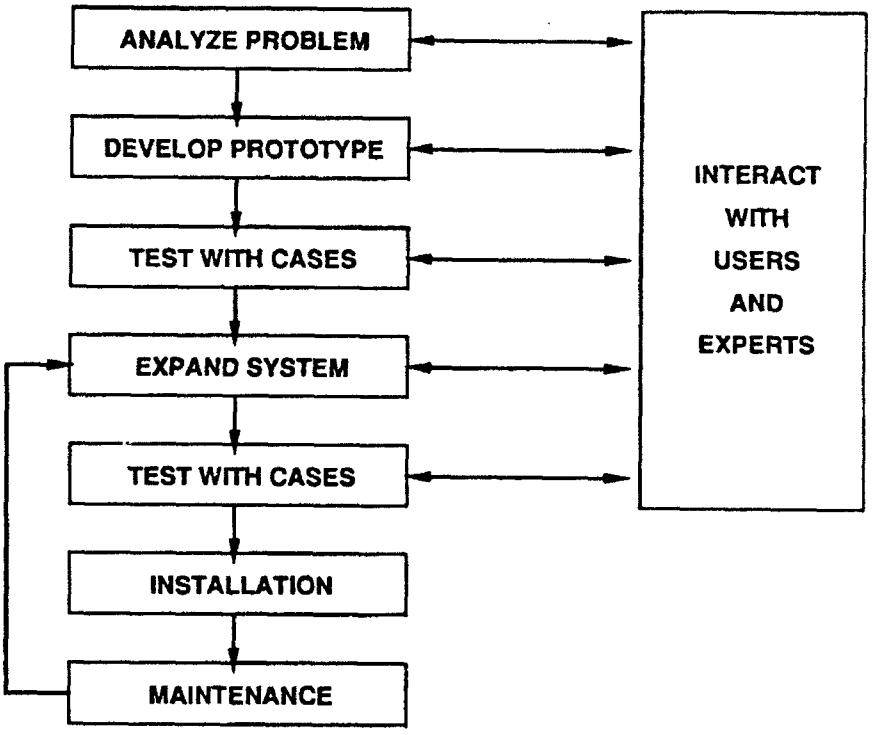


Figure 2-3: Harmon et al.'s (1988) Representation of the Traditional Expert System Development Methodology

In the past, expert system developers have been prone to taking a "we versus they" attitude when comparing their system development approach to the more conventional one. However, as we mentioned in Chapter 1, that is beginning to change because, for all its strengths, expert system development efforts emphasizing a totally "experimental" prototyping approach have not been as successful as we would like to believe. There have been too many failures because expert system developers have failed to consider the requirements issues of critical concern to sponsors. To quote Cholawsky (1988, p. 44), "In general, prototypes ignore both deployment issues (such as cost-benefit analysis, scaling up to operational size, and handling real-world data) and transition issues. ... The development team argues that business issues should be temporarily tabled; if the problem cannot be solved technically, it does not matter if it is justified from a business sense. This argument has a fatal flaw. Even if the life underwriting decisions of the

expert system exactly match the underwriter, the system will not be built if it lacks an adequate payback." Furthermore, Constantine and Ulvila (in press) have found a number of cases where the development cycle was to "prototype forever," never reaching an operational system.

As we mentioned earlier, a more requirements-driven development process is evolving. This process emphasizes the importance of prototype planning that explicitly identifies objectives and evaluation criteria for determining prototype success prior to development as a means of keeping development on track. In addition, it emphasizes conventional software design activities, not just knowledge engineering. These design activities provide a structured approach for addressing, during prototype development, many of the deployment and transition issues that have been a problem for successful prototype implementation.

Figure 2-4 presents Cholawsky's (1988, p. 47) "new approach to expert system development." Her approach divides prototyping activities into two groups. The first group emphasizes prototype planning. It includes specifying the objectives and secondary issues (i.e., subproblems) for the prototype, the evaluation criteria for "determin[ing] prototype success," and a development schedule with milestones and deliverables. The second group emphasizes prototype development. It includes a predesign stage for understanding the domain vocabulary, a logical architectural design stage for analyzing the reasoning and representation paradigms used in the domain, a physical architectural design stage for considering hardware and software issues, an implementation stage for programming the knowledge engineered during the logical architectural design stage, and an evaluation stage for explicitly testing the prototype against the evaluation criteria specified during planning. Iteration is assumed throughout the various stages, although it is more controlled than in the traditional prototyping approach. Assuming that a successful prototype is developed, efforts are then directed toward developing the operational system, and maintaining and enhancing it.

Cholawsky is not alone in proposing an expert system development approach that moves toward integrating aspects of the more conventional system development approach into prototyping. Figure 2-5 presents Weitzel and

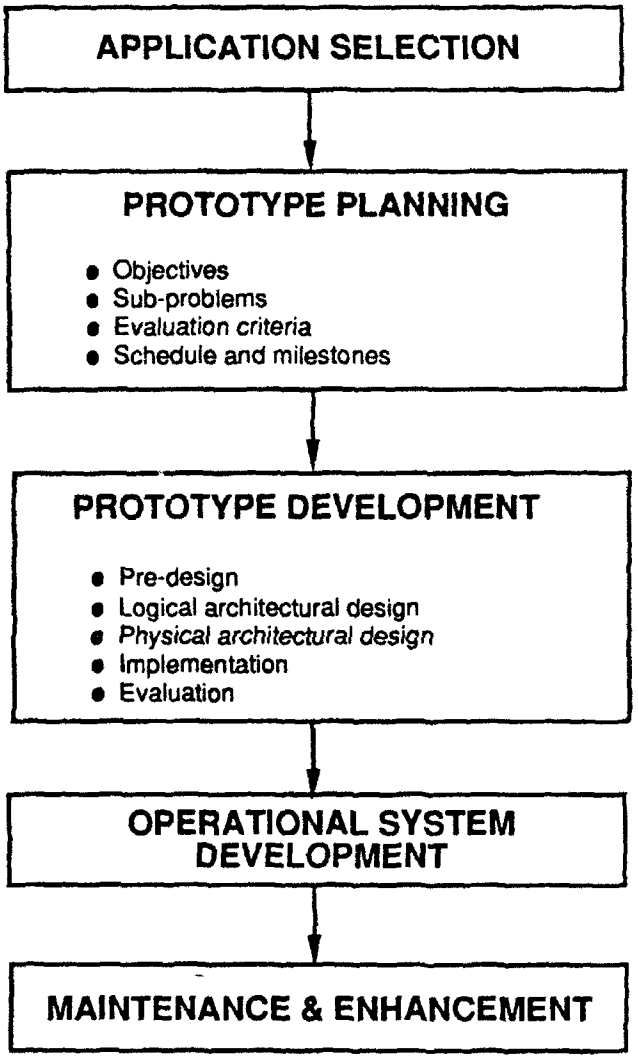


Figure 2-4: Cholawsky's (1988) "New" Approach to Expert System Development

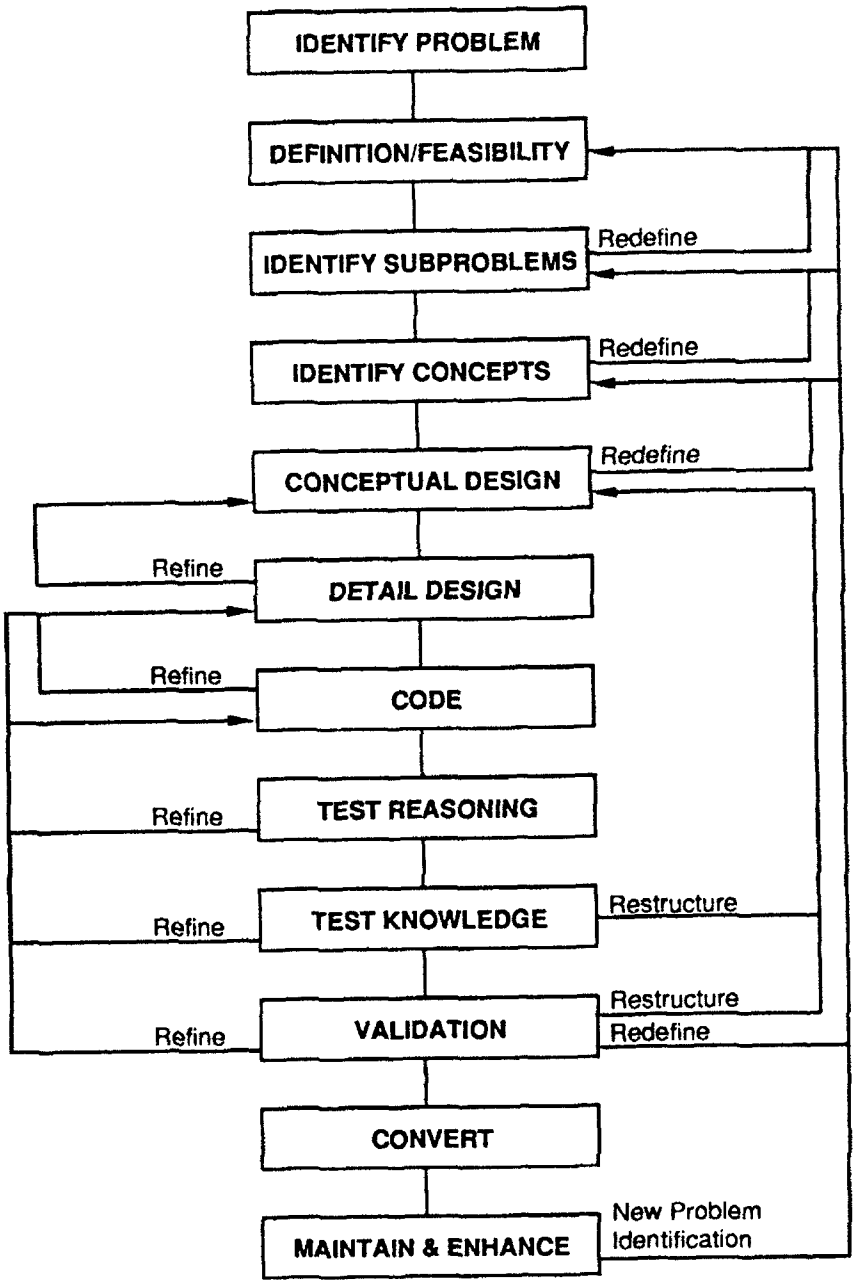


Figure 2-5: Weitzel and Kerschberg's (1989) Representation of the "Knowledge-Based System Development Methodology Flow"

Kerschberg's (1989, p. 599) "knowledge-based system development methodology flow." Although their approach emphasizes iteration, it also emphasizes conceptual and detailed design prior to coding, as well as substantial testing and evaluation. Figure 2-6 presents Rook and Croghan's (1989, p. 589) "knowledge acquisition activity matrix." As can be seen, they have tried to integrate the various knowledge-engineering activities with the steps in the conventional system development cycle in an effort to move effectively and efficiently from the laboratory to the operational environment. And Figure 2-7 presents Andriole's (1989, p. 31) "prototyping design blueprint." Although his approach emphasizes modeling and iteration, it also emphasizes aspects of the more conventional system development approach, particularly requirements analysis, hardware and software selection, and system design, packaging, transfer, and evaluation.

Figure 2-8 presents Wolfgram et al.'s (1987, p. 17) "stages of expert system development." This representation is quite similar conceptually to Cholawsky's (1988). First, Wolfgram et al. also distinguish between prototype planning and development. Planning issues, such as the specification of goals (i.e., objectives and subproblems), evaluation criteria, and explicit requirements for guiding development, are part of Identification and Definition, the first stage in their development approach. Second, prototype development and construction of the operational version of the expert system are distinctly different stages of development—stages 2 and 3, respectively. Moreover, prototype development in Wolfgram et al's approach incorporates many of the same requirements and design issues Cholawsky addresses in her approach. All of the issues are directed toward designing the "structure" for the operational version of the system.

Regardless of the various representations of the expert system development approach, test and evaluation is an inherent aspect of it. In fact, test and evaluation activities are assumed; they are simply taken for granted as part of iterative development. Consider Wolfgram et al. (1987, p. 19), for example. "Once the prototype is in place, it is a working model, or submodel, of the planned complete expert system. It is at this stage that, after careful testing and review, a decision is made whether to continue the project and construct the complete expert system or abandon the project." Testing and

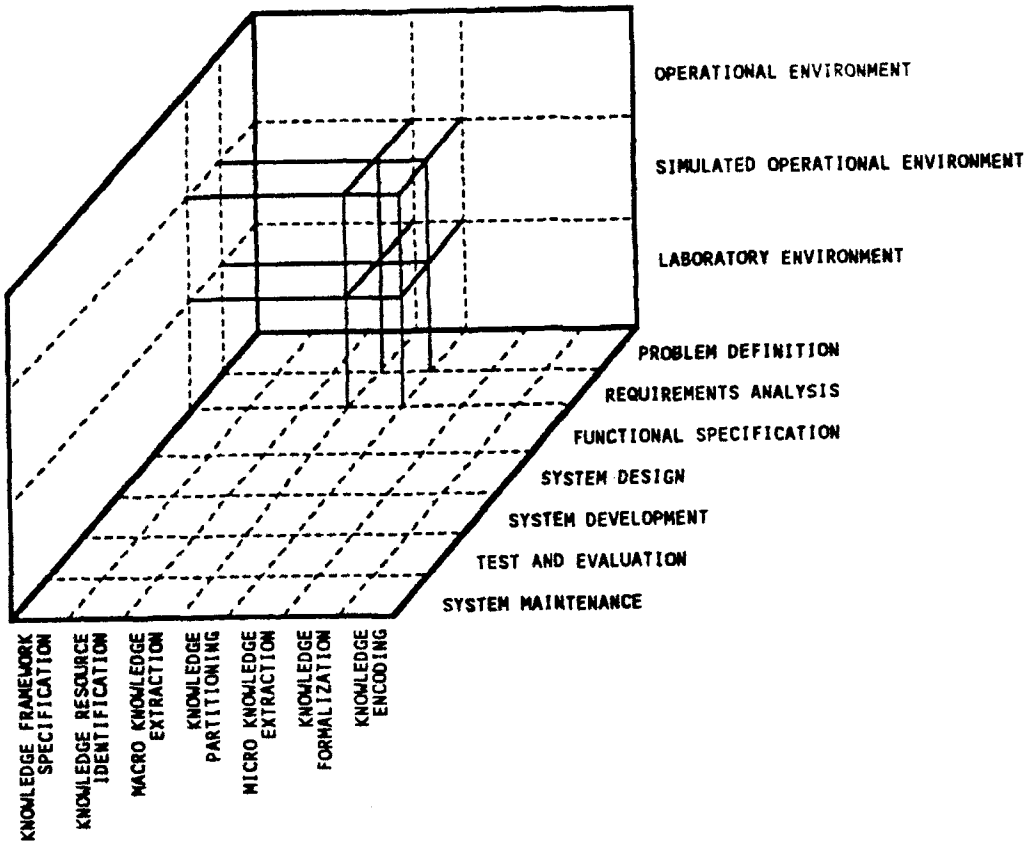


Figure 2-6: Rook and Croghan's (1989) "Knowledge Acquisition Activity Matrix"

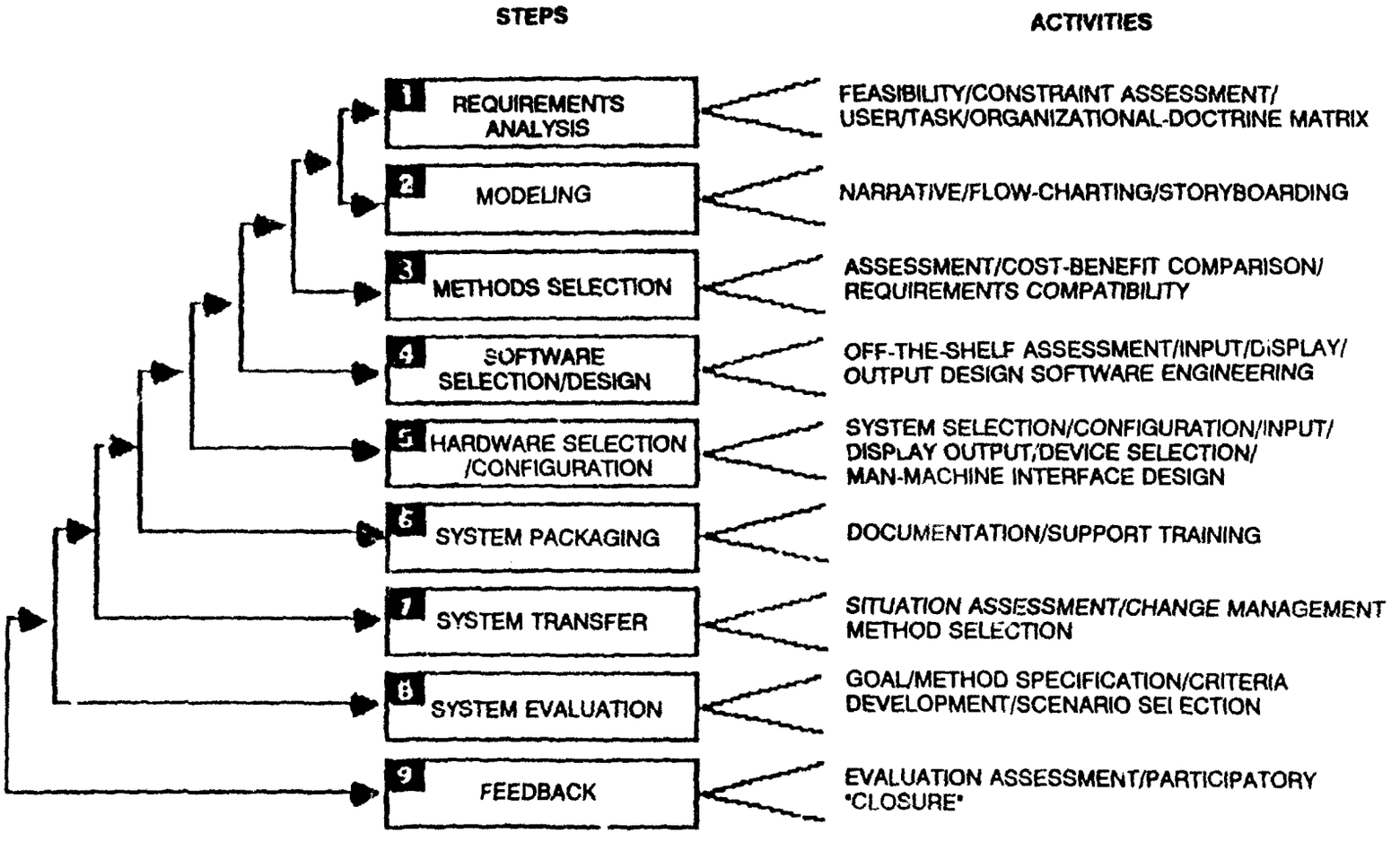


Figure 2-7: Andriole's Nine-Step Prototyping Design Blueprint

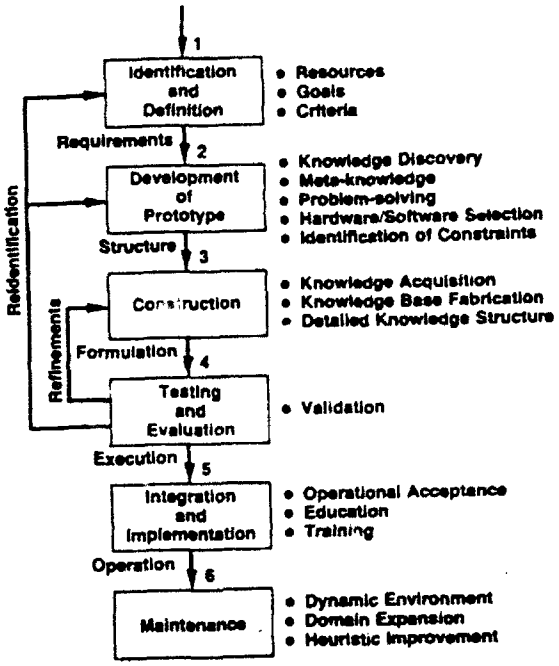


Figure 2-8: Wolfgram et al.'s (1987) "Stages of Expert System Development"

evaluation are the critical activities upon which the fate of the project rests, yet Wolfgram et al. fail to indicate its presence in their pictorial representation of the development process.

Figure 2-9 presents a revised version of Wolfgram et al.'s representation of the expert system development process, but now with testing and evaluation explicitly added to the process. In particular, we have added test and evaluation boxes after prototype development (stage 2), integration and implementation (stage 5), and maintenance (stage 6). Testing and evaluation obviously occurs at the end of each of these development stages. For example, in addition to testing and evaluating the prototype, one would obviously test and evaluate the operational version of the expert system (stage 5); one does not simply hand it over to the host organization and walk away from it after so much time and money has been spent in developing it. Similarly, one tests the effects of any changes that one makes to the system during maintenance for fear that an enhancement might result in an unanticipated error or problem

All this test and evaluation goes on informally in most expert system development efforts. Indeed, informal test and evaluation is a pervasive activity in development. Webster's dictionary (1966) uses the word "examine" as part of its definition of both "test" and "evaluate." As developers, we are always examining the system. We're always trying to find things here and fix problems there in order to improve our product.

The reason we have added the test and evaluation boxes to the development approach is to formalize that activity. Moreover, good test and evaluation is not epitomized by the informal examination of the system. It is epitomized by the use of explicit and appropriate methods for helping members of the development and sponsoring teams make the numerous judgments and decisions inherent in expert system development. Remember, test and evaluation represents the control mechanism for providing the feedback that keeps the development effort on track. This point is clearly illustrated in Andriole's (1989) prototyping design blueprint, Figure 2-7. The ultimate goal of test and evaluation is to help senior-level decision makers in an organization decide whether the option of developing and implementing an expert system, either singularly or in combination with other actions, is an effective organization-

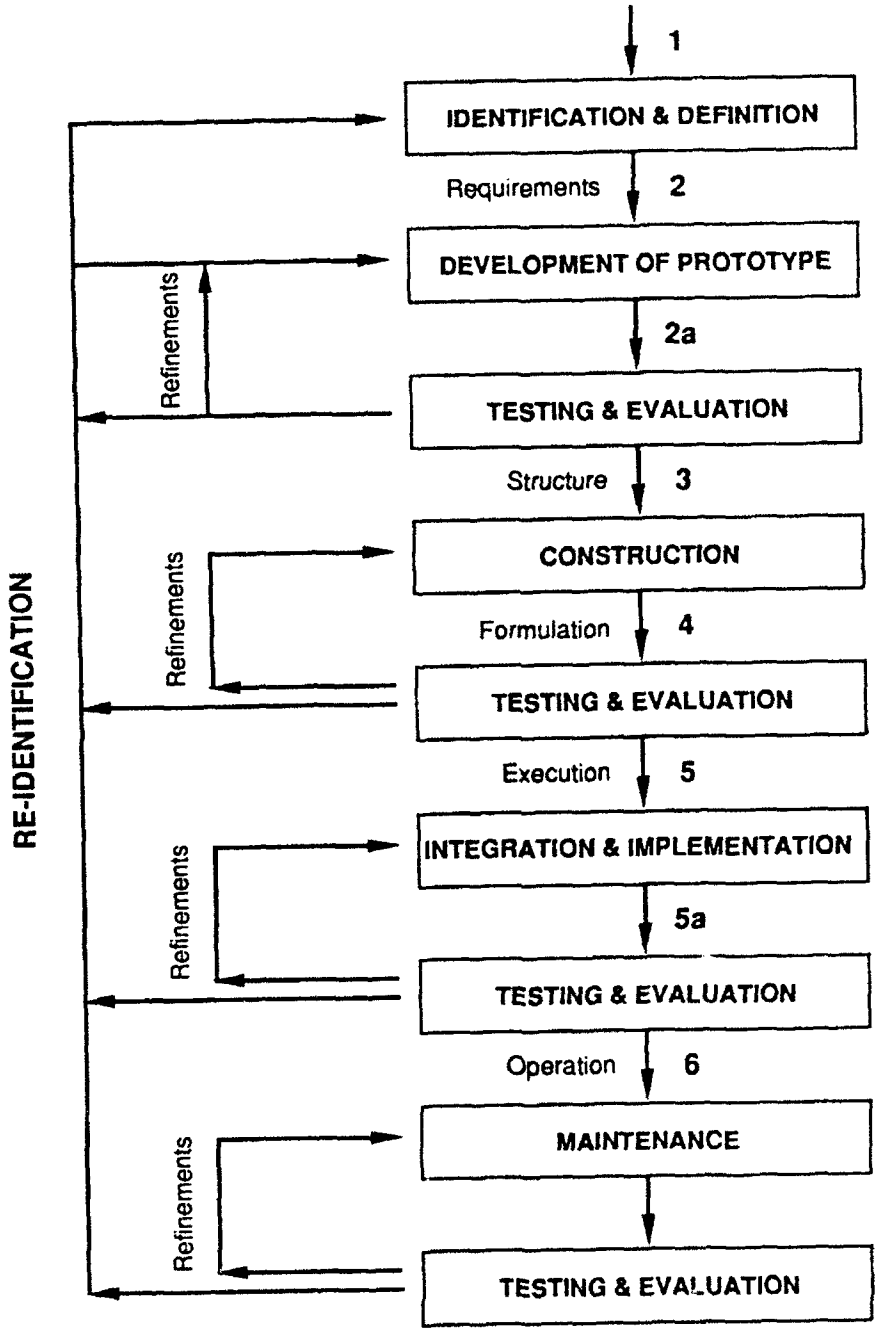


Figure 2-9: Modification of Wolfgram et al.'s Representation in Order to Emphasize Test and Evaluation

al response for dealing with their present or future problem environment. Once the development process is underway, the application of formal test and evaluation methods permit one to monitor the perceived utility of the expert system under development, and take corrective action to increase the probability of its use and effectiveness.

At this point the reader may be thinking that emphasizing formal tests and evaluations will increase development costs. In fact, it might. Formal testing and evaluation is an expensive process. Although we do not have data for expert systems, Hetzel (1984) points out that direct testing costs for major software systems approach 25 percent of the development costs. Direct testing costs include reviews, program testing, systems testing, acceptance testing, test planning and design, computer time, and test resources, both human and material. This is obviously not a trivial investment. To quote Gould and Lewis (1985, p. 306), "... testing still has a price. It is nowhere nearly as high as commonly supposed, however, and it is a mistake to imagine that one can save by not paying this price. ... If it is not done in the developer's lab, it will be done in the customer's office."

The failure to systematically test and evaluate a system during its development often results in "indirect costs," as Hetzel (p. 174) calls them. Indirect costs include "rewriting programs, recovery, corrective action costs, rekeying data, failures, analysis meetings, debugging, retesting," etc. "Indirect testing costs, or the costs of poor testing, are usually at least twice the direct costs and may be spectacularly higher." Moreover, indirect testing costs are substantially more expensive later in development. For example, empirical research (e.g., Rushby, 1988) indicates that errors due to faulty requirements are between ten to one hundred times more expensive to fix if detected during implementation than during requirements analysis. Given our track record, there is no reason to assume that these estimates are any different for expert systems. Of course, these costs pale by comparison to the potential costs of a catastrophic decision or even the costs of a system that is ignored or unused because of correctable problems that could have been detected by testing. All of this suggests that formally incorporating test and evaluation into development is a wise investment.

SUBJECTIVE, TECHNICAL, AND EMPIRICAL TEST AND EVALUATION METHODS: AN OVERVIEW

We now overview the various subjective, technical, and empirical test and evaluation methods. The methods will be discussed only at a general level here. More detailed discussions and illustrations will be presented in later chapters.

Subjective Test and Evaluation Methods

The goal of subjective tests and evaluations is to assess the expert system from the perspective of potential users and sponsors. This is accomplished by identifying measures of effectiveness (MOEs) that will provide the information required to assess the system's utility. The explicit identification of MOEs is particularly important at the beginning of the development process because they represent (a) reference points for the development team to use, and (b) criteria for evaluators to monitor in order to assess whether the development process is on track.

Gaschnig et al. (1983, p. 258) have emphasized the importance of developing MOEs early in the expert system development process. "It is important for system designers to be clear about the nature of their motivations for building an expert system. The long-range goals must also be outlined explicitly. *Thus stage 1 of a system's development, the initial design, should be accompanied by explicit statements of what the measures of the program's success will be and how failure or success will be evaluated* [italics theirs]. It is not uncommon for system designers to ignore this issue at the outset, since the initial challenges appear so great upon consideration of the decision-making task that their expert system will have to undertake. If the evaluation stages and long-range goals are explicitly stated, however, they will necessarily have an impact on the early design of the expert system."

Multiattribute Utility Assessment. Riedel and Pitz (1986, p. 986), as well as others (e.g., Adelman and Donnell, 1986; Andriole, 1989; Keeney and Raiffa, 1976; Ulvila et al., 1987), have pointed out that multiattribute utility assessment (MAUA) "... provides a formal structure for conceptualizing

MOEs, a mechanism for both decomposing the global MOE into its component dimensions and for reintegrating them to yield one summary measure of value." When applying MAUA to the evaluation of expert systems and other types of DSS, the system is conceptually decomposed into attributes that can be defined well enough so that one can obtain either subjective or objective measures (MOEs) of how well the system performs on each attribute. This decomposition typically proceeds through the creation of a value hierarchy, such that the global attribute entitled "the overall utility" is decomposed into major categories of attributes, which are further decomposed, and so forth, until one is reasonably confident that one can define and obtain precise, reliable, and valid measures (or scores) of the system on each attribute. Table 2-1 presents the MAUA value hierarchy developed by Adelman and Ulvila (in press).

Reintegration typically occurs within MAUA through the application of utility functions and relative importance weights. An expert system is usually evaluated on many different attributes, all of which need to be defined as precisely as possible. The natural measurement scale for an attribute depends on the nature of the attribute. For example, the scale for an attribute could be in objective units (e.g., minutes for time) or subjective units (e.g., how strongly a subject likes a feature) depending on the attribute. Nevertheless, a common scale is required to compare scores on one attribute with scores on another—that is, "apples with oranges"—and, by so doing, obtain an overall score for the system.

A utility scale, which conceptually measures psychological value or satisfaction, meets this requirement. Utility (or value) functions are used to translate system performance on an attribute into a utility score on that attribute. Then, relative importance weights (or other forms of decision rules) are used to assess the relative value of a utility score on one attribute with the utility score on another and, thereby, obtain an overall utility score for the system. [This weighting procedure is formally valid if additivity assumptions are met; see Keeney and Raiffa (1976). An assumption of additivity is generally a reasonable approximation; see Edwards (1977).]

MAUA was used to provide the subjective evaluation of DART described in Chapter 1. As you will remember, "decomposition" was illustrated in Table 1-

1, which presented a multiattributed hierarchy that decomposed the global MOE (the overall utility of the expert system) into three component dimensions: the user/expert system, user-system/decision making organization, and organization/environment interfaces. Each of these three interfaces or branches in the MAUA hierarchy, were further decomposed into bottom-level attributes (or MOEs).

"Reintegration" was achieved by a three-step procedure. First, the experts completed a questionnaire that essentially scored DART on each of the bottom-level attributes. Second, we assumed a positive linear utility function for each bottom-level attribute, thereby conceptually converting the performance score on the attribute into a utility score on that attribute. And, third, we used equal weights moving up the hierarchy to combine the (utility) scores for lower-level attributes into more global scores at the next level of the hierarchy until we obtained an overall score on the global MOE.

The Dollar-Equivalent Technique. The dollar-equivalent method is a means for translating all benefits, as well as costs, into dollar values instead of utilities, as in MAUA. In the dollar-equivalent method, all benefits are converted into dollar equivalents by "pricing out." Pricing out is a judgmental technique that is much the same as the procedure for converting performance scores into utility scores in MAUA. As Huber (1980, p. 83) points out, the dollar-equivalent method is "... a special case of the more general ... MAU model technique." Consequently, the appropriateness of the method depends on the defensibility of the conversions to monetary equivalents. Relatedly, traditional cost-benefit analysis represents all benefits and costs in dollars, and then uses the ratio of benefits to costs as the basis for decision making.

Decision Tree Analysis. Decision tree analysis is a formal method for combining uncertainties, which are represented as probabilities, with utilities when evaluating alternative decision options. Decision tree analysis often uses subjective probabilities of scenarios to represent, at a collective level, the uncertainties inherent in the decision-making situation facing members of the sponsoring team. These scenarios represent the members'

hypotheses regarding alternative states of the world, a perspective that is perfectly consistent with the SHOR paradigm. The overall expected utility of different organizational options, including whether or not to develop an expert system, depends on the (a) probabilities assigned to the various scenarios, and (b) the utility of each of the options for each of the scenarios.

Other Subjective Test and Evaluation Methods. A fourth subjective method is a MAUA-based cost-benefit analysis that uses optimization procedures to identify the set of options that provides the greatest utility at specific (total) levels of cost (see Ulvila and Chinnis, in press). As Adelman (1990b) pointed out, this approach is particularly appropriate when the funding horizon is uncertain for identifying the best (a) set (or suite) of decision support technology (and/or other organizational options), and (b) configuration of components for a particular system at different levels of dollar cost. For example, Rockmore et al. (1982) used this approach to select DART and four other decision support systems for enhancing U.S. Air Force tactical decision making. However, to the best of our knowledge, this subjective method has not been used to evaluate potential expert systems.

Although we will not describe them here, there are other subjective evaluation methods that have been used to test and evaluate expert systems. For example, Liebowitz (1986) has used the Analytical Hierarchy Process developed by Saaty (1980), Tong et al. (1987) have proposed a frame-based approach, and Klein and Brezovic (1988) and Slagle and Wick (1988) have used subjective test and evaluation approaches analogous to MAUA. The interested reader is urged to consider them.

Discussion. An important characteristic in common among the subjective test and evaluation methods described herein is that they develop an analytical model to represent the judgments of the participating decision makers. One of the principal advantages of a "model" is that it permits sensitivity analysis; members of the sponsoring team can change their judgments and see if the changes have any effect on the results. For example, does changing the relative importance placed on an attribute (e.g., response time) in a MAUA suggest that a different alternative design for the expert system be imple-

mented? This is an important capability early in the expert system development process because, consistent with the SHOR paradigm, there may exist considerable interpersonal disagreement among members of the sponsoring team due to both information input uncertainty regarding the hypotheses and consequence-of-action uncertainty regarding options.

Moreover, both the MOEs and methods used to convert performance measures into MOEs developed early in the development process will be used during and after development to evaluate the prototypes and final expert system, respectively. All that will change with time is basis for these judgments, both in terms of the specificity of the option (i.e., the expert system) and, for certain MOEs, the availability of empirical and technical performance data. Consequently, it is important to obtain consensus among the sponsoring team, which, it is assumed here, includes representative user(s) of the expert system, early in (if not prior to) the development process. To quote O'Connor (1989, p. 103), "These attribute trade-offs are not after-the-fact evaluation issues. Rather, they are decision issues relevant to the design problem and should be resolved before detailed system design and testing take place."

There is a long line of research (for a review, see Delbecq et al., 1975) demonstrating that, more often than not, structured facilitation procedures can focus a group's discussion, thereby increasing the probability, not only of a more accurate final position, but one that is more strongly supported by the group. The subjective methods described above further improve discussion by letting members of the sponsoring team focus on a quantitative model instead of each other. Eils and John (1980), for example, found that groups using MAUA procedures in conjunction with group facilitation procedures tended to make more accurate decisions than groups using only facilitation procedures.

Better discussion occurs because group members have to define their thought processes in order to provide the numerical inputs required by the model. At the same time, however, the model permits group members to retreat from their original position, or more strongly voice it, on the basis of the numerical outputs and sensitivity analyses. Directing the discussion toward aspects of the model helps remove some of the "personal" focus of group

decision making. As was mentioned earlier, explicit identification of MOEs and procedures for converting performance scores into a global MOE, represent reference points for the development team to use when developing the expert system, and criteria for the evaluator to monitor in order to assess whether the development process is on track.

The above discussion has focused on the applicability of subjective test and evaluation methods early in development in order to define (a) what the expert system has to be capable of doing in order for the decision maker who is using it to consider it to be a good system, and (b) whether development of such an expert system is feasible given the financial, time, personnel, and other constraints operating in the situation. However, subjective test and evaluation methods are applicable throughout the development effort. The reader should remember that subjective methods like MAUA, cost-benefit analysis, decision analysis, and the different variations on these themes were all developed to help decision makers systematically evaluate decision options, regardless of what they might be. Consequently, they are potentially applicable anywhere in the development process where members of the development team need to evaluate one option against another. In those areas important enough to warrant their use, they represent an audit trail for indicating why one action was taken versus another.

Finally, we have tried to stress the importance of using subjective test and evaluation methods to evaluate whether the prototype(s) and final, operational version of the expert system are consistent with the initial goals and objectives of the sponsoring team. It is important to point out that objectives, and particularly the tradeoffs among them, can change during the course of the development process either because of the changing environment with which the sponsoring team is dealing, changes in the membership of the sponsoring team, the insights gained during the development process regarding what is technically feasible/infeasible, etc. Subjective evaluation methods provide an effective mechanism for representing these changes and, through sensitivity analysis, estimating their implications for the development process.

Technical Test and Evaluation Methods

Three classes of technical evaluation methods are, in turn, briefly overviewed in this chapter: (a) static testing for assessing the logical consistency and adequacy of the knowledge base; (b) using domain experts for assessing the functional completeness and predictive accuracy of the knowledge base; and (c) conventional software test and verification methods for assessing the service requirements of the entire system.

Logical Consistency and Completeness. As Rushby (1988) points out, the concepts of static testing in conventional software testing can be readily extended to expert systems because, in both instances, the focus is on detecting anomalies in the program without actually executing it on test cases. To quote Rushby (p. 92), "An anomaly in a program is nothing more than an apparent conflict between one indication of intent or purpose and another ..." The types of anomalies of particular interest in expert systems pertain to the logical consistency and logical completeness of the knowledge base.

Researchers (e.g., Kirk and Murray, 1988; Nazareth, 1989; and Rushby, 1988) have developed taxonomies of anomalies in the knowledge base that are amenable to static testing. Some of these anomalies are listed below. In doing so, we assume that the knowledge base is represented in the form of "if-then" production rules or can be transformed into such a representation. As Nazareth (1989, p. 257) points out, "For systems that employ more involved representation schemes, the nature of the verification task may differ." (However, Hayes (1981) has shown the consistency between rules and frames, which indicates that similar concepts are applicable to frame-based knowledge representation.)

- *Redundant Rules.* Individual rules or groups of rules that essentially have the same conditions and conclusions.
- *Subsumed Rules.* When one rule's (or rule group's) meaning is already expressed in another's that reaches the same conclusion from similar but less restrictive conditions.
- *Conflicting Rules.* Rules (or groups of rules) that use the same (or very similar) conditions, but result in different conclusions.

or rules whose combination violates principles of logic (e.g., transitivity).

- *Circular Rules*. Rules that lead one back to an initial (or intermediate) condition(s) instead of a conclusion.
- *Unnecessary If Conditions*. Values on a condition that do not affect the conclusion of any rule.
- *Unreferenced Attribute Values*. Values on a condition that are not defined; consequently, their occurrence cannot result in a conclusion.
- *Illegal Attribute Values*. Values on a condition that are outside the acceptable set of values for that condition.
- *Unreachable Conclusion* (and *Dead Ends*). Rules that do not connect input conditions with output conclusions.

Static testing for the above anomalies could be performed manually for small, well-structured knowledge bases. For even moderately sized knowledge bases, however, this approach is precluded by the amount of effort required and the probability of disagreements among testers. Consequently, researchers (e.g., Culbert and Savely, 1988; Franklin et al., 1988; Nguyen et al., 1987; Stachowitz et al., 1988) have begun developing automated static testers. We do not have the space here to discuss these different efforts. However, we do want to caution the reader that automated static testers are not without their limitations. To quote Nazareth (1989, pp. 265-266), "In most cases the verification process is closely dependent on the structure of the problem domain, making translation of principles to other systems difficult. Additionally, only a subset of the errors identified [above] are covered. ... The expansion of verification scope has serious implications for detection. ... [And] the majority are directed toward applications without uncertain inference." Nevertheless, automated static testers represent a major step forward in assessing the logical consistency and completeness of a knowledge base. Unfortunately, such static testers are not available commercially, nor are there plans to make static testers available in the near future.

Functional Completeness and Predictive Accuracy. By functional completeness we mean to address the range of domain-oriented questions, such as whether the knowledge base contains all desired input conditions and output conclusions, or even "knows" its knowledge limitations. Some of these

questions can be answered by domain references. However, the level of domain expertise typically desired for expert systems is typically not codified in such references. Indeed, Davis (1989) has argued that one of the major contributions of expert system technology is the organization and codification impacts it has on various disciplines. Consequently, domain experts are typically required to evaluate the functional completeness of the system. However, one should remember that the system's level of functional completeness depends on its stage of development and, most importantly, the domain requirements resulting from the requirements analysis (step 1).

The predictive accuracy of the knowledge base pertains to the correctness by which the rules (or whatever representation scheme) relates input conditions to output conclusions. Such an assessment is essential for expert systems, for "garbage in" is literally "garbage out." Consequently, experts, both those who participated in development and particularly those acting as independent evaluators, should be used to evaluate the predictive accuracy, and thus adequacy, of the knowledge base. Expert evaluation typically proceeds in two ways: through examination of the knowledge base and the evaluation of test cases.

Expert examination of the knowledge base typically focuses on whether the system exhibits "correct reasoning." The obvious concern is, of course, that the knowledge base not have mistakes. However, another concern, and one which Gaschnig et al. (1983) pointed out is not shared by all developers, is whether their programs reach decisions like human experts do. Many psychologists have long argued that this concern can not be answered for one cannot look inside an expert's head to obtain the "correct reasoning." Instead, all one can do is build "paramorphic models" (Hoffman, 1960) of the reasoning process, and evaluate their predictive accuracy against test cases. Indeed, researchers (e.g., Dawes and Corrigan, 1974; Einhorn and Hogarth, 1975; Levi, 1989; Stewart et al., 1988) have shown that simple linear models can often result in prediction as good as that achieved by the far more complex models found in expert systems, or even by the experts themselves.

As Lehner and Adelman (in press) point out in their review of the literature, this is not a resolved issue. To quote Gaschnig et al. (1983, p.

255), "... there is an increasing realization that expert-level performance may require heightened attention to the mechanisms by which human experts actually solve the problems for which the expert systems are typically built." In addition, Adelman, Rook, and Lehner (1985) found that domain experts' judgments of the utility of decision support system (including expert system) prototypes were significantly affected by the match between how they and the system attempted to solve the problem. This suggests that, at a minimum, the system's representation and presentation scheme needs to be reviewed. However, if the principal objective is to develop a system that maximizes predictive performance, then simple linear models, or mathematical models unrepresentative of how the experts solve the problem, may be more appropriate than models of human experts in certain situations.

The predictive accuracy of the knowledge base is assessed using test cases and performance standards. The desired standard is ground truth; that is, the unambiguously correct answers to the test cases. Correct answers are most desirable because substantial research (e.g., see Ebert and Kruse, 1978; Goldberg, 1970; Yu et al., 1979) has shown that experts do not always make perfect inferences and, in fact, often disagree with one another in the kinds of complex domains for which many expert systems are developed. Often, but not always, it is inappropriate to expect better predictive accuracy from the system than the expert. (This may not be the case where the system incorporates knowledge from a limited, well-defined domain—such as a procedure manual—or where the system represents the expertise of several experts. Here, it may be appropriate to expect the system to be more accurate than any given expert. Also, Brian Smith points out that "we already ask machines to do things that people don't do," such as land an airplane in fog, and that in many serious applications the standard of doing as well as a human is not good enough (Davis, 1989).)

If ground truth measures exist, one can try to discriminate between "accuracy" and "bias" in a signal detection sense (Lehner, 1989). Accuracy refers to the degree of overlap in the distributions of belief values when the hypothesis is true versus false. Bias refers to the proportion of false negatives (hypothesis true, but user says false) to false positives (hypothesis false, but user says true).

If the correct answers do not exist or, for whatever reason, are inappropriate for the test cases, then one must rely on the judgment of an expert or the consensus judgment of a group of experts. Considerable care must be given to structuring the experts' activities. In particular, the evaluation team must ensure that the experts are "blind" as to whether the system or other experts generated the conclusions to the test cases. This is typically referred to as a "Turing test" (e.g., see Rushby, 1988).

In closing this subsection, it is important to note that test case construction is an important issue. To quote O'Keefe et al. (1987, p. 83), "The issue is not the *number* of test cases, it is the *coverage* of test cases—that is, how well they reflect the input domain. The input domain is the population of permissible input ..." [italics theirs]. The required coverage capabilities is clearly a statement that needs to be a result of the requirements analysis. For as O'Keefe et al. point out, developers frequently devote a disproportionate amount of time to attempting to ensure that the system can handle the truly "expert" cases that may occur very infrequently. Moreover, these "infrequent" cases often become the test cases. This may or may not be appropriate depending on the requirements for the system, and it can certainly be expensive.

An alternative identified by O'Keefe et al. is to randomly select test cases using a stratified sampling scheme such that the relative frequency of the cases is representative of those in the operational environment or stipulated in the requirements. Additionally, test cases should be chosen to cover situations where a failure in the system would be especially serious. It is also important that some of the test cases simulate the most common operation of the system. Finally, Lehner and Ulvila (1989) have shown that the number and type of test cases depend on the level of expert system performance that users consider to be valuable. The greater the difference between the average levels of predictive accuracy with (versus without) the system, considered necessary by users, the smaller the number of test cases actually required to test whether the expert system meets the criterion requirement. This point will be considered in substantial detail in Chapter 5.

Service Requirements. Verification testing should be systematically performed for the service requirements of expert systems, just like any other software product. Fagan and Miller (as reported in DeMillo et al., 1987) have identified four phases for software testing. The first phase is manual analysis in which the requirements specification and design and implementation plan are analyzed for problems by experienced software engineers. The second phase is static analysis, which may be manual or automated, in which requirements and design documents and software are analyzed, but without code execution. The third phase is dynamic analysis in which software is executed with a set of test data, such as in random testing, functional testing, and path testing. The fourth phase, which Fagan and Miller consider to be optional, is attempting to prove the program as being correct, such as in mathematical verification. Detailed discussions of these and other methods can be found in, for example, DeMillo et al. (1987), Fairley (1985), Pressman (1982), and Rushby (1988).

Discussion. In closing this subsection, we want to make four points about technical evaluation methods. First, as Hamlet (1988, p. 666) points out, each method has its strengths and weaknesses and therefore, represents "imperfect test methods." Therefore, testers need to use multiple methods to obtain accurate feedback. Second, the intent of testing is to find errors. As Fairley (1985, p. 268) points out, "... one has most confidence in programs with no detected bugs after thorough testing and least confidence in a program with a long history of fixes." Third, the best way to minimize the number of errors and the amount of time, effort, and money required to fix them, is to eliminate errors early in development. Consequently, as Gelperin and Hetzel (1988) point out, software development life cycles are becoming "preventive" through the application of software testing methods early in the development process. And, fourth, testing methods using experts to evaluate the knowledge base rely heavily on empirical analysis via test data. However, the reader should keep a clear distinction between the empirical results of technical and empirical evaluation methods. The former focus on how well the expert system's knowledge base was developed; the latter focus on how much better system users, who may not be experts, can perform the task using the expert system.

Empirical Test and Evaluation Methods

Empirical evaluation methods can be classified into experiments, quasi-experiments, case studies, simulations, and statistical analyses of historical data (e.g., see Adelman, 1990b). Only the first two methods are considered here.

Experiments. Experiments are, by far, the most common and commonly thought of empirical evaluation method. Moreover, they are particularly appropriate when a number of people would actually use the developed expert system, for experiments are designed to help generalize from a test sample to the larger population.

One typically thinks of two kinds of experiments—benchmark testing and factorial designs. The first kind tests the system against objective benchmarks that represent performance constraints. If the system passes the benchmarks, it proceeds further; if it fails, it undergoes further development or is set aside. "For example, it is not enough to know that with the aid the user can arrive at a decision in 30 min[utes]. If the organizational user required a decision in 30 min[utes], the aid would be effective. If a decision was needed in 15 min[utes], the aid would not be effective" (Riedel and Pitz, 1986, pp. 984-985).

It should be noted that such performance benchmarks differ from the more traditional time and efficiency measures used to benchmark computer systems. [Note: Readers interested in the latter are referred to Press (1989), who benchmarked different expert systems on the time required to load and execute different types of knowledge bases, and the amount of disk space required in source and fast-load formats.] Both classes of benchmarks typically get developed during requirements analyses emphasizing a features-based approach. Although such performance constraints may be necessary in real-time, life-critical activities, they are unnecessary for many expert system applications.

Performance benchmarks represent noncompensatory decision rules; that is, the system's other features do not compensate for failing the performance benchmark. Such a position may be inconsistent with the decision rule guiding

the sponsoring team's evaluation. For example, it's quite possible that the sponsoring team would give up some time for task accomplishment in order to gain an improvement on other MOEs, such as decision performance.

The second kind of experiment is a factorial design (e.g., see Cochran and Cox, 1957) where (a) one or more factors are systematically varied as the independent variables, and (b) the dependent variables are quantitative, objective measures of system performance. There are five basic components of factorial experiments. First, there are the participants, or subjects, in the experiment. These may or may not be experts depending on the targeted users of the expert system's advice. We focus on "users" because the system operators may or may not be the actual decision makers.

Second, there is the task that the participants perform during the experiment. Test cases are often embedded in larger scenarios representative of the organization's problem-solving environment in order to effectively assess (1) the users' ability to solve problems with and without the system, and (2) their opinion of system characteristics, such as its speed, explanation capabilities, organizational fit, etc. Remember, the expert system may be addressing only part of a much larger organizational decision.

Third, there are the experimental conditions or independent variables of interest, such as whether the participants perform the task with or without the expert system. The level of task difficulty should be either as representative of the operational environment as possible or matched to the required performance capabilities of the system. The capabilities of the system depend on its stage of development (e.g., see Gaschnig et al., 1983; Marcot, 1987).

Fourth, there are the dependent variables (or MOEs) of interest. Objective measures (e.g., performance and speed), observational measures (e.g., regarding how the system is used) and subjective measures (e.g., user confidence in the solution) can all be used as dependent variables. In the case of decision quality, one should use either ground truth measures (i.e., the correct answer) for the task or, if they do not exist or are inappropriate, the consensus or collective judgment of experts. If ground truth measures exist, one should discriminate between "accuracy" and "bias" in a

signal detection sense, as was done for the knowledge base. If experts are used, "blind" ratings as to which experimental conditions produced the solutions are again required to control against bias. Using at least two experts who have not participated in the development is advocated here because of the substantial empirical research showing expert disagreement. However, the use of one expert is acceptable if the requirement is that the expert system emulate the judgments of that expert.

Fifth, there are the procedures governing the overall implementation of the experiment. Substantial care should be directed toward accurately representing the unaided as well as aided condition to ensure a fair test. If performance is better in the "aided" condition, we want to be able to say that it is due to the expert system and not some other extraneous factor. In order to do so, we need to (ideally) try to control for all "plausible rival hypotheses" (Campbell and Stanley, 1966, p. 36) that might explain the obtained findings. Toward that goal we introduce the concepts of reliability and validity.

Yin (1984, p. 36) defines reliability as "demonstrating that the operations of a study—such as the data collection procedures—can be repeated, with the same results." The key concept is replication. In contrast, "valid" is defined by Webster's dictionary (1966) as that which is sound because it is "well grounded on principles or evidence." If an experiment is valid, its conclusions can be accepted; that is, rival hypotheses have been controlled for.

An experiment can be reliable, but its conclusions invalid. However, an experiment cannot be valid if it is unreliable; that is, one cannot conclude that the results are well grounded if the evidence upon which they are based is undependable. The basis for good experimentation is, therefore, reliable (i.e., dependable) procedures and measures. Although far from trivial, reliability is typically possible in experimentation because of high experimenter control. For example, the experimenter can pilot-test and subsequently modify the procedures and measures until they produce the same result when applied to the same situation, regardless of who performs the experiment

We consider four types of validity. First, Yin (1984, p. 36) has defined internal validity as "establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships." As Cook and Campbell (1979, p. 38) note, "Internal validity has nothing to do with the abstract labeling of a presumed cause or effect; rather, it deals with the *relationship* between the research operations *irrespective of what they theoretically represent*" [italics theirs]. Although there are numerous threats to internal validity, randomization of participants to experimental conditions is the most effective means for guarding against them.

In addition, one needs to consider the experiment's construct validity, its statistical conclusion validity, and its external validity. Yin (1984, p. 36) has defined construct validity as "... establishing good operational measures for the concepts being studied." Construct validity is required in order to "make generalizations about higher-order constructs from research operations" (Cook and Campbell, 1979, p. 38) in a particular study. Good construct validity means that we are measuring that, and only that, which we want to be measuring. Of particular concern in expert system evaluations is that the "system treatment" is not confounded with something else. If confounding exists, then the "something else" represents rival hypotheses that could explain our obtained results.

"Statistical conclusion validity is concerned not with sources of systematic bias but with sources of random error and with the appropriate use of statistics and statistical tests" (Cook and Campbell, 1979, p. 80). The former concern is with whether the study is sensitive enough to permit reasonable statements regarding the covariation between the independent and dependent variables. The latter concern is with what constitutes appropriate statistical tests of these statements. We will return to both concerns in substantial detail in later chapters.

As Campbell and Stanley (1966, p. 5) point out, "*External validity* asks the question of *generalizability*: To what populations, settings, treatment variables, and measurement variables can this effect be generalized?" [italics theirs]. Within the context of expert system evaluations, external validity

deals with the extent to which the results of an experiment conducted in a simulated (laboratory) setting will generalize to an operational environment. Consistent with an iterative, prototyping approach, the representativeness of the experimental setting and the level of the system's performance requirements should advance throughout the development cycle. Although the latter is routinely acknowledged, the former is not. It must be remembered that expert systems and, indeed, most information and decision technology, fail to be successfully implemented for organizational, not technical, reasons. Consequently, increasing the fidelity of the organizational and environmental interfaces between the system and its users is essential in generalizing the performance results obtained in the laboratory to the real world.

Quasi-Experiments. Ideally, field experimentation would be used to assess if the expert system significantly improved performance in an actual organizational setting. For example, appropriate organizational units (e.g., sections in a company or governmental agency) would be randomly assigned to the "with system" and "without system" conditions, and their performance measured until it stabilized. If possible, a "placebo" condition would be included too. Organizational units in this condition would be given some "treatment" that was not hypothesized to have any effect on performance. This is analogous to giving patients sugar pills when evaluating new drugs, and is oriented to controlling for the "Hawthorne effect" (e.g., see Schein, 1970) confounding in the "with system" condition that is the result of being given special treatment and not the technology. The unit of analysis is the performance of the organizational unit; consequently, a large enough sample of units would be required for performing statistical tests.

The sample size and randomization requirements of true experiments is typically not possible in many organizations. Quasi-experimental designs should be used in such situations. To quote Campbell and Stanley (1966, p. 34), "There are many social settings in which the research person can introduce something like experimental design into his scheduling of data collection procedures (e.g., the *when* and *to whom* of measurement), even though he lacks the full control over the scheduling of experimental stimuli (the *when* and *whom* of exposure and the ability to randomize exposures) which make a true

experiment possible. Collectively, such situations can be regarded as quasi-experimental designs" [italics theirs].

There are a number of different types of quasi-experimental designs. Among the ten types identified by Campbell and Stanley (1966) are: (a) time series designs, where the organizational unit would be measured for a long period of time before and after receiving the system; (b) multiple time series designs that do not use randomization, but do use a control group that does not receive the system; and (c) nonequivalent (and nonrandomized) control group designs that rely on statistical techniques like analysis of covariance to assess whether the pre-test and post-test difference for the expert system group is significantly better than that of the control group. These and other empirical test and evaluation methods will be considered in greater detail in later chapters.

CHAPTER SUMMARY

This chapter had three principal sections. The first section overviewed test and evaluation criteria identified by Adelman and Ulvila (in press). Although the specific criteria one would use would depend on the specific requirements of one's users and sponsors, the criteria presented herein contained the wide range of test and evaluation criteria commonly found in the literature and, therefore, can give one a broad list of criteria from which to start. The second section of the chapter overviewed the expert system development approach. Although this approach is moving toward incorporating aspects of more traditional software systems engineering, it is still epitomized by iteration, prototyping, and test and evaluation. And in the third section, we overviewed the many different types of subjective, technical, and empirical test and evaluation methods. A multi-faceted test and evaluation is required in order to provide the different kinds of information that developers and sponsors need in order to assess the utility of an expert system, both during and after development.

All three classes of test and evaluation methods are applicable during a formal test and evaluation of an expert system prototype by an outside group, as was shown in Chapter 1. In addition, however, specific methods are more or

less applicable at other times in the development cycle. In particular, subjective evaluation methods are applicable early in the cycle because they represent an explicit means for defining the judgments of members of the sponsoring team and potential users of the system.

Technical test and evaluation methods are also applicable during design and development. For example, as part of the knowledge elicitation and representation process, one should routinely assess the adequacy and accuracy of the knowledge base by using (1) static testing to help assess the knowledge base's logical consistency and completeness, and (2) experts, both those participating in development and those acting as evaluators, to help assess the knowledge base's functional completeness and predictive accuracy. In addition, traditional software test and verification methods can be used to help assess the "service" versus "competency" requirements of the expert system.

In contrast to technical test and evaluation methods, which focus on how well the system was developed, empirical test and evaluation methods focus on how well decision makers can perform their task(s) with (versus without) the system. From an iterative, prototyping perspective, it is anticipated that experiments will be conducted throughout development as a means of objectively measuring the performance of the expert system and testing hypotheses for improving it. After transferring the expert system to the test organization, experiments, quasi-experiments, and case studies can be used to evaluate performance in the actual, organizational setting.

The different methods overviewed herein address the different test and evaluation criteria represented in the hierarchy shown in Table 2-1. This hierarchy not only represents a framework for summarizing the criteria, but for integrating them using multiattribute utility assessment. In particular, we will demonstrate that this hierarchy can be used in conjunction with MAUA scoring and weighting procedures to assess the overall utility of an expert system to users and sponsors. Consequently, with these thoughts in mind, we now turn to consider subjective test and evaluation methods in more detail the next chapter.

CHAPTER 3:

MORE ABOUT SUBJECTIVE TEST AND EVALUATION METHODS

The last chapter overviewed five different subjective evaluation methods: multiattribute utility assessment (MAUA), cost-benefit analysis, the dollar-equivalent technique, decision analysis, and a MAUA-based cost-benefit analysis. We also briefly overviewed how feature-based criteria lists and value of information analysis, two other subjective evaluation methods, can be subsumed under the broader methods of MAUA and decision analysis, respectively. This chapter will (1) overview the applicability of each of the five subjective methods in more detail, and (2) provide details of a specific MAUA-based method for testing expert systems.

This chapter will continue to emphasize the importance of using subjective evaluation methods to link together Steps 1 (requirements analysis) and 8 (evaluation) in the expert system development process through the feedback provided in Step 9 (feedback). For it is by defining, at the outset of the development effort, the requirements for evaluating the expert system that helps ensure that the development effort will stay on track and that the expert system will be used by the persons for whom it is being developed. Moreover, the sponsoring team's objectives, and particularly the tradeoffs between objectives, can change during the course of the development effort, either because of the changing environment with which the sponsoring team is dealing, changes in the membership of the sponsoring team, the insights gained during the development process regarding what is technically feasible or most appropriate, etc. Subjective evaluation methods provide an effective mechanism for representing these changes and, through sensitivity analysis, estimating their implications on the global measure of effectiveness (MOE). More generally, such methods provide an explicit mechanism (and audit trail) for evaluating whether the prototypes and final, operational version of the expert system are consistent with the sponsoring team's goals and objectives.

The applicability of subjective evaluation methods to the other steps in the development process will not be emphasized in this chapter. As was pointed out in the last chapter, these methods can be readily used by the

development team throughout the development process—for example, in formulating specific requirements, in evaluating off-the-shelf software (e.g., shells) versus project-specific software, or in evaluating various hardware configurations. Remember, these subjective methods and variations were all developed to help decision makers evaluate systematically decision options, regardless of what they might be. This presentation is an adaptation of the methods for testing expert systems. The methods are applicable anywhere in the development process where members of the development team need to evaluate one option against another. Thus, in those areas important enough to warrant their use, they represent an audit trail for indicating why one action was selected over another.

Finally, it is important to again point out that Andriole (1989) identifies a wide range of "requirements analysis methods" and taxonomies for profiling the task, user, and organizational requirements. These methods include open- and closed-ended questionnaires, various types of interviewing procedures, the observation of users' behavior as they perform scenarios (i.e., hypothetical decision problems), protocol analyses where users describe their decision-making processes as they perform scenarios, etc. We will not discuss these methods because they are concerned with requirements analysis rather than testing and evaluation. (For the same reason we will not discuss prototyping methods or systems engineering methods.)

We do want to point out, as Andriole does, that requirements analysis methods are fallible; consequently, members of the development team should use multiple methods in order to ensure the reliability and validity of the results of the requirements analysis. By reliable we mean that the same method will produce the same results at different times. By valid we mean that the results are, in fact, related to the utility of the expert system. Consistent with the prototyping strategy, we would expect less reliability and validity of the results early in the development process. As a tester or evaluator, you can help the development team assess which aspects of the requirements analysis need more work, as well as what other methods could be used to improve the analysis, before moving on to develop the functional

model of the system. In this respect, you should find subjective evaluation methods particularly helpful.

MULTIATTRIBUTE UTILITY ASSESSMENT (MAUA)

There are numerous texts (e.g., Huber, 1980; Keeney and Raiffa, 1976; Pitz and McKillip, 1984) and papers (e.g., Edwards, 1977; Einhorn and McCoach, 1977) describing MAUA. As Huber (1980, p. 46) has pointed out, "Multiattribute utility models (MAU models) are designed to obtain the utility of items or alternatives that have more than one valuable attribute; therefore, they must be evaluated on more than one criterion. A MAU model essentially shows a decision maker how to aggregate the utility or satisfaction derived from each of the various attributes into a single measure of the overall utility of the multiattributed item or alternative." Expert systems are clearly "items" that have numerous attributes (or characteristics) of potential value to a decision maker. MAUA represents a method for combining how well an expert system scores on these attributes (i.e., individual measures of effectiveness) into an overall assessment.

All of the subjective evaluation methods that we will consider in this book proceed by a "divide and conquer" or "decomposition and reintegration" approach. When applying MAUA to the testing and evaluation of expert systems, the expert system is conceptually decomposed into criteria that can be defined well enough so that one can obtain either subjective or objective measures of how well the expert system performs on each of them. This decomposition typically proceeds through the creation of a value hierarchy, such that the global criterion entitled "the overall utility (or value) of the expert system" is decomposed into major categories of criteria (e.g., the knowledge base, the inference engine, etc. as shown in Table 2-1). These categories are further decomposed, and so forth, until one is reasonably confident that one can define and obtain precise, reliable, and valid measures (or scores) of the expert system on each bottom-level criterion in the hierarchy.

[Note: According to Huber (1980), the bottom-level criteria should be called "attributes." This convention is not strictly adhered to and, in

fact, it is not uncommon to use the words "criteria" and "attributes" interchangeably. Moreover, it is not uncommon for the names Multiattribute Utility Assessment (or Analysis), Multiattribute Utility Theory, and Multicriterion Decision Making to be used synonymously, even though purists within each "variation on the theme" might take issue with this state of affairs. In this book, we will try to consistently use the term "attributes" to refer to the bottom-level evaluation criteria. However, the reader should not be concerned if the terms are used synonymously.]

By "precise" one means that the attribute's definition is sufficiently clear and unambiguous so that everyone knows exactly what characteristic of the expert system is being measured by the attribute and how to measure it. By "reliable" one means that, at a minimum, one will get approximately the same score for an expert system on an attribute if one uses the same measurement instrument at two different points in time. This is referred to as "test-retest" reliability. The measurement instrument could be subjective (e.g., a person's score in answering a question) or objective (e.g., a performance score in an experiment). In addition, one would hope to obtain "inter-instrument" reliability as well, such that two measures of an attribute, whether they are subjective or objective, would produce approximately the same scores. Finally, by "valid" one means that the attribute is, in fact, related (or contributes) to the overall utility of the expert system as determined by the key decision maker or the sponsoring team. While many people would like to think that objective, performance scores are the only valid MOEs, the overall decision regarding the value of an expert system is invariably a mix of subjective and objective measures.

More broadly, it is desirable that the MAU hierarchy have the following general features: be (1) comprehensive enough to account for all the different MOEs deemed important when evaluating the expert system; (2) capable of differentiating between an acceptable and unacceptable (or "good" vs. "bad") system; and (3) composed of independent attributes. Although the first two features appear clear and straightforward, the last one may appear counter-intuitive and it is not absolutely essential. To quote Ulvila et al. (1987, p. 25), "While it is desirable to satisfy the last characteristic, it is by no means required. It is possible to define

evaluation factors that are dependent upon each other and interact in complex ways. However, most of the value of an MAU model can usually be obtained by using a simpler form in which each factor is independent of all other factors. If it is clear that two factors are not independent, but both are interacting, it is sometimes possible to define a single factor that incorporates the critical aspects of the dependent factors. (Notice that here we are addressing independence in the *worth* [italics theirs] of an attribute, not technical independence—e.g., run time, computer usage, performance speed, and judgmentally assessed speed are likely to be highly dependent but may represent attributes of separate interest to the tester.)"

It is important to note that a hierarchy, while extremely helpful, is not absolutely essential. All that is essential is that one be able to define a comprehensive set of (independent) attributes that can be measured precisely, reliably, and validly so that the overall utility score can differentiate between an acceptable and unacceptable expert system. The hierarchy simply helps one perform this task.

The application of MAUA during, or even prior to, the requirements analysis step is typically oriented toward helping the sponsoring and development teams (a) identify the broad organizational requirements the expert system needs to satisfy, and (b) select the general type of expert system that will satisfy these requirements. The hierarchy of MOEs presented in Table 2-1 does provide a comprehensive reference point (or checklist) of requirements that an expert system should satisfy and, therefore, provides an effective design and evaluation tool for guiding and monitoring, respectively, the ongoing development process. It does not, however, necessarily provide an effective hierarchy of MOEs for initially selecting the general type of expert system to develop, for that decision may require different types of information.

As a tester or evaluator, the hierarchy of MOEs and, more generally, the application of MAUA should be tailored to the objectives and information needs of the members of the sponsoring team with which one is working. In fact, few structuring techniques have been proposed by

decision scientists. Two techniques have, however, been routinely used by analysts applying MAUA. a top-down (or hierarchical) approach (Keeney and Raiffa, 1976), and a bottom-up (or attribute listing) approach (Kelly, 1973). The top-down approach to structuring the hierarchy proceeds as follows: the upper level nodes are listed first; then each node, in turn, is subdivided into its component attributes. The process continues until it identifies the lowest-level attributes. In contrast, the bottom-up approach proceeds by first obtaining a list of all of the possible attributes (i.e., lower-level nodes of the hierarchy) without any concern for their hierarchical arrangement. An effective procedure for doing this that is quite consistent with the SHOR paradigm is to ask the participant (e.g., decision maker) to describe how the alternatives are different (e.g., better or worse) from each other. The specific differences typically represent the bottom-level attributes. The attributes are subsequently clustered together to form the criteria representing the branches of the hierarchy.

In general, there has been very little research evaluating the relative effectiveness of MAUA structuring techniques, and only one study (Adelman, Sticha, and Donnell, 1986) doing so under controlled, experimental conditions where there existed an accepted multiattributed hierarchy as an external criterion for measuring effectiveness. With regard to the latter, Adelman et al. found no significant difference in the accuracy of top-down and bottom-up structuring techniques. Although equivocal, their results did, however, indicate that the top-down technique results in deeper hierarchies than the bottom-up technique. Since the deeper hierarchies did not result in more accurate ones, these results suggest that greater depth is merely a by-product of the top-down approach and not a function of a more comprehensive problem decomposition. Their "post hoc" analysis strongly suggested that combining the two approaches would result in significantly more accurate hierarchies. The hierarchy of expert system attributes described later in this chapter was developed by a combination of top-down and bottom-up techniques.

Reintegration typically occurs within MAUA through the application of utility functions and assessment of relative importance weights. Remember

the expert system is being evaluated on many different attributes. The natural measurement scale for an attribute depends on the nature of the attribute. For example, the scale for an attribute could be in objective units (e.g., minutes for time), subjective units (e.g., the eleven-point questionnaire scale used in the DART evaluation), or categories (e.g., yes or no to the presence of a feature), depending on the nature of the attribute. A common scale is, however, required in order to compare scores on one attribute with scores on another (i.e., combining "apples with oranges") and, by so doing, obtain an overall assessment for the item (e.g., expert system) being evaluated by the decision maker. A "utility" scale, which conceptually measures psychological value or worth or satisfaction, meets this requirement. Utility (or value) functions are used to translate the performance on an attribute into a utility score on that attribute. Then, relative importance weights (or other forms of combination rules) are used to assess the relative value of a utility score on one attribute with the utility score on another.

Utility functions for individual attributes tend to be linear, increasing or decreasing in form, as we used in the case study shown in Chapter 1. But as Hammond et al. (1975) point out, there is no reason why they can not be U-shaped or inverted U-shaped or even a step-function such that the utility score on an attribute is zero until a certain level of performance is achieved on the attribute. The functions are represented pictorially by utility curves, such as the hypothetical ones shown in Figure 3-1 from Ulvila et al. (1987, p. 29).

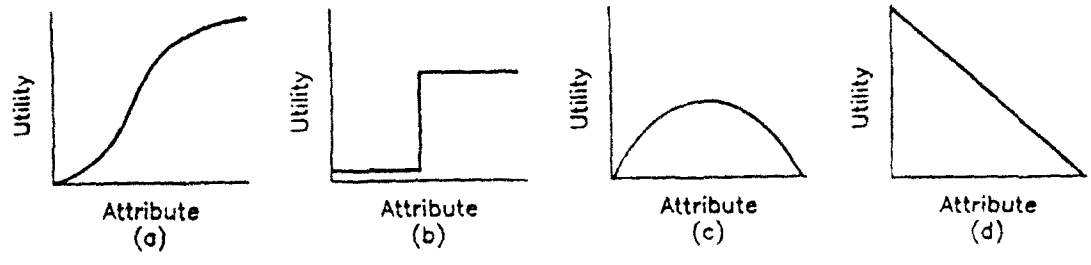


Figure 3-1: Some Possible Shape Utility Functions

The specific range for the utility scale is arbitrary; for example, Huber (1980) uses a 0 to 100 range throughout his book and Keeney and Raiffa (1976) use a 0 to 1.0 range throughout theirs. What is critical, however, is the relative utility (or value) of the scores on the scale, and the relationship between differences on the scale. A utility score of 50 on a 0 to 100 utility scale (or 0.5 on a 0 to 1.0 utility scale) indicates that it is mid-way in value between the lowest and highest values on that scale. The difference between 25 and 50 on a utility scale is equivalent to the difference between 50 and 75 on that scale. The actual values on the natural scale for the attribute that corresponds to these utility values will, more often than not, fail to correspond to such a straight-line function.

Consider the hypothetical utility function shown in Figure 3-2 from Ulvila et al. (1987, p. 29), which transforms the time required to set up an artificial intelligence system into a utility score in the military context they were considering. A utility score of 100 is obtained for a set-up time of 0 minutes; a utility score of 0 is obtained for a set-up time of 60 minutes. One obtains half (or more) of the utility if the system is set up in 5 minutes (or less). Moreover, an increase from 5 to 15 minutes, which has a utility scale value of 25, was considered as serious as an increase from 15 to 60 minutes. This utility scale is clearly telling the developer the importance of a fast set-up time to the user.

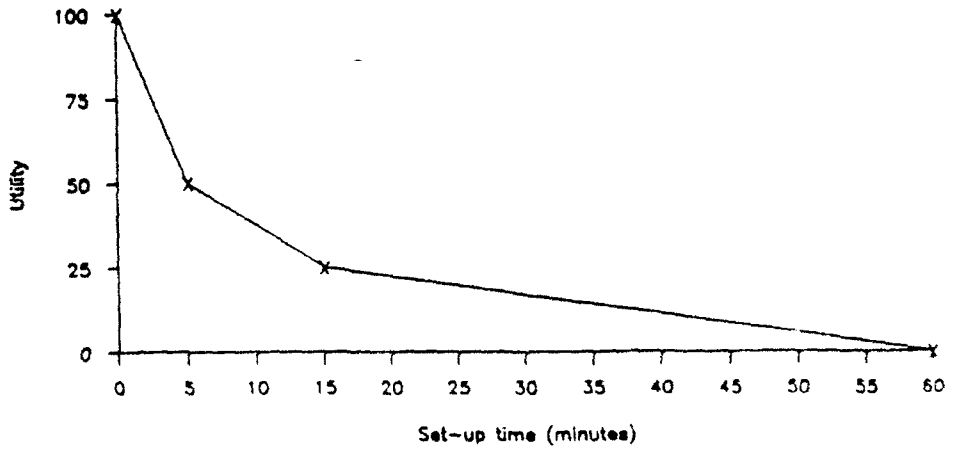
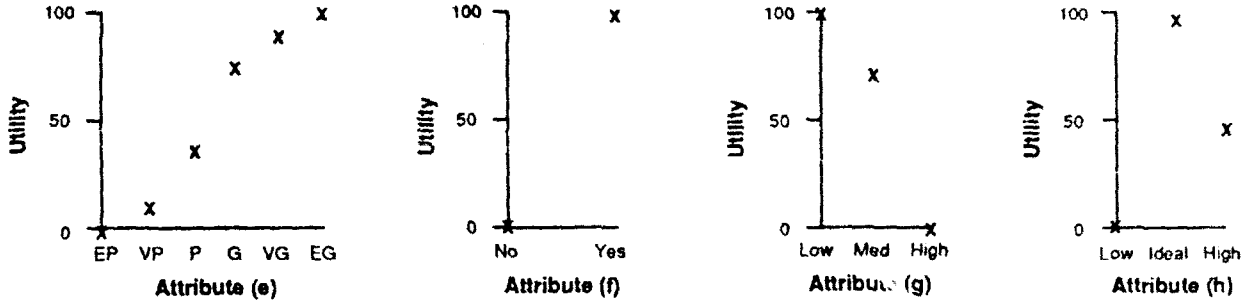


Figure 3-2: Hypothetical Utility Function for Expert System Set-Up Time

Nor are utility functions limited to characteristics with continuous measures. Utility functions can also be constructed for categorical variables or other variables with discrete units. Some examples are shown in Figure 3-3. The important features are that the horizontal axis uniquely determines the state of the attribute, and the vertical axis specifies the value of the states.



EP = Extremely Poor
VP = Very Poor
P = Poor
G = Good
VG = Very Good
EG = Extremely Good

Figure 3-3: *Possible Discrete Utility Functions*

But how important is the relative importance of one attribute versus another? The relative importance of a utility score on a bottom-level attribute is reflected typically by (1) its relative weight compared to the other bottom-level attributes comprising a component, and (2) the relative weight of the components moving up the hierarchy. For example, Figure 3-4 (from Buede and Adelman, 1987, p. 143) considers the relative importance of five attributes, which we'll initially assume are all bottom-level attributes to the same upper-level criterion. Specifically, each of the five rectangles in the top half of Figure 3-4 represents the utility scales for an attribute. The rank order of the rectangles (going from left to right) represents the rank order of the attributes in terms of their relative importance; that is, attribute A is more important that attribute B, and so forth. The relative height of the rectangles indicates their relative importance weights. For example, attribute B is about 60% as tall as

attribute A; consequently, a utility score of 100 on attribute B is equivalent to a utility score of 60 on attribute A. Similarly, attribute C is half as tall as attribute B; consequently, a utility score of 100 on attribute C is equivalent to a utility score of 50 on attribute B and a utility score of 30 on attribute A. A score of 50 on attribute C is equivalent to a score of 25 on attribute B and a score of 15 on attribute A.

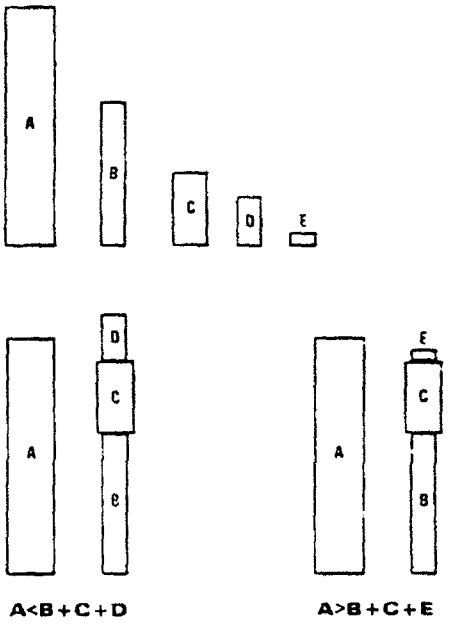


Figure 3-4: A Pictorial Representation of the Relative Importance of Different Utility Scales

The bottom half of Figure 3-4 illustrates the "paired comparison" weighting technique, which utilizes the (utility) scaling concepts illustrated in the top half of the figure. Specifically, it shows that a utility score of 100 on attribute B plus a utility score of 100 on attribute C results in a (combined) utility score of only 90 on attribute A; consequently, the combined relative importance weights for attributes B and C must be less than the weight for attribute A. In the example shown, the added importance weights for attributes B, C, and D are greater than that for attribute A, but the added weights for attributes B, C, and E are not. By comparing the overall value of a utility score of 100 on each of the

attributes, one is able to assess the relative importance of the attributes.

The same procedures can be used to assign relative weights to the attributes at the next level of the hierarchy, and so forth up the hierarchy until all the attributes at each level of the hierarchy have been assigned relative importance weights. The weights at each level of the hierarchy should be proportional such that the sum of the weights at each level is the same. We recommend scaling the weights to sum to 1.0 at each level so that the overall utility scale is on the same scale as is being used for each attribute. If one then multiplies the weights along each branch from the top to the bottom of the hierarchy, one will obtain a cumulative weight on each bottom-level attribute that indicates the overall importance of one bottom-level attribute versus another.

This method of assigning weights to attributes assumes that the attributes are "additively independent" (Keeney and Raiffa, 1976). Roughly speaking, additive independence is a condition where the utility for improvements in one attribute does not depend on the levels of the other attributes. Other, more complicated formulations are possible, and many of them are described by Keeney and Raiffa (1976). However, Edwards (1977) notes that (p. 250), "theory, simulation computations, and experience all suggest that [the additively independent form] yields extremely close approximations to very much more complicated "true" utility functions, while remaining far easier to elicit and understand." The additive form is assumed in the framework described later in this chapter and used throughout this book.

As has been discussed thus far, reintegration of the bottom-level scores for an expert system into the assessment is achieved in MAUA by the weighted sum of all the utility scores. This can be represented algebraically by equation [3-1]:

$$U(i) = \sum_j w_j u(x_{ij}) \qquad [3\text{-}1]$$

where:

$U(i)$ is the overall utility for alternative i;

w_j is the "cumulative" relative weight on attribute (j);

$u(x_{ij})$ is the utility value for alternative i on attribute j; and

Σ indicates the summation over all attributes.

Equation [3-1] focuses on the bottom-level attributes in a hierarchy, for the relative weights (w_j) in Equation [3-1] represent the "cumulative weights" on the bottom-level attributes. They are obtained by multiplying the weights along each branch of the hierarchy from the top to the bottom.

The same numerical results can be obtained if one goes from the bottom up in the hierarchy. That is, one would multiply the noncumulative relative weights and utility scale values achieved by the alternative for each of the bottom-level attributes in the hierarchy. One would obtain a score for the criterion at the next higher level of the hierarchy by summing the weighted utility scores for all the bottom-level attributes that it comprises. The process is then repeated. One would multiply this score by the relative weight for the criterion to obtain a weighted score for the criterion. Then, one would add the weighted scores to obtain a utility score for the criterion category at the next higher level of the hierarchy, and so forth, moving up the hierarchy until one obtained an overall utility score.

As Hogarth (1987) has pointed out, the additive decision rule shown in Equation [3-1] is a compensatory combination rule because high utility values on certain attributes can compensate for low values on other attributes and still result in a good score on the global MOE. However, as Riedel and Pitz (1986) pointed out, it might be more appropriate to use a noncompensatory rule to ensure that the expert system gets a low score on the global MOE if it fails to achieve the necessary performance level on a critical bottom-level attribute. This perspective that can be readily handled arithmetically in MAUA by using (1) a zero/one utility score to reflect whether or not the expert system passed the threshold on the

critical dimension(s), and (2) a multiplicative combination rule to obtain the global MOE utility score. Alternatively, we recommend the use of thresholds for attributes that are noncompensatory. Thresholds should be set for attributes where nonperformance on the attribute should lead to a poor overall assessment regardless of the performance on other attributes. Using this system to evaluate an expert system, a failure to pass a threshold is noted for all attributes where the failure occurs, and this notation is carried up in higher-level assessments regardless of the system's weighted-average utility score. This threshold system is utilized in the MAUA computer program described by Ulvila et al. (1987).

In closing this discussion, it is important to emphasize that MAUA can be used to create an assessment structure for combining an expert system using both objective and subjective MOEs. Its application, however, might initially be disturbing to (and difficult for) members of both the sponsoring and developing teams, for it emphasizes the subjective process decision makers typically go through when evaluating expert systems. To quote Riedel and Pitz (1986, pp 987-988), "The [utility scales] and weights are necessarily personal judgments by the decision maker that express the contribution each attribute makes to the overall MOE. There is no way to avoid the fact that the overall MOE must be based on such judgments, or the fact that no mechanical procedure can replace this subjective assessment ..." This does not mean, of course, that MAUA is the only subjective evaluation method that one can use to evaluate how well an expert system is meeting the sponsoring team's requirements, but it will be the major subjective method used in this book. In particular, we propose a MAUA framework later in this chapter, and we propose weights based on characteristics of the expert system in Chapter 7. We now turn to consider a second subjective method that has been used for system evaluation—cost-benefit analysis.

COST-BENEFIT ANALYSIS AND THE DOLLAR-EQUIVALENT TECHNIQUE

As Riedel and Pitz (1986, p.991) point out, "In making decisions about a system, cost is often an important factor ... The problem is how to integrate the cost factor into the evaluation design." With MAUA, cost is

simply considered as one of the (higher-level) MOEs. Its impact on the evaluation is determined by its impact on the overall utility score, which is achieved by (a) the utility function translating dollar costs into a utility score, and (b) the relative importance given to the cost MOE. As Huber (1980, pp. 79- 83) points out, in traditional cost-benefit analysis and the dollar-equivalent methods, however, all the benefits, as well as costs, are translated into dollar values instead of utilities. In the former, standard economic or accounting practices, such as employing the rate of return or time value of money concept, are used to create monetary equivalents. In the latter, "... the monetary equivalents are developed judgmentally when the standard economic techniques are stretched beyond their limits."

The perhaps surprising conceptual similarity between cost-benefit analysis and MAUA can be illustrated by listing the following five principal steps for implementing the former, as identified by Keim and Janaro (1982): (1) identification of pertinent measures of effectiveness, that is, benefits; (2) the description of alternatives; (3) the "expression" of performance and cost as functions of the characteristics of each alternative; (4) the estimation of appropriate (dollar) values for the (performance) equation parameters; and (5) the computation, sensitivity analysis, and presentation of results. This sounds remarkably like the MAUA procedures described above where one (a) decomposed the global MOE into a hierarchy of MOEs (i.e., attributes); (b) defined the alternatives; (c) identified the natural scale value for each bottom-level attribute and obtained the scores for the alternatives on scales; (d) constructed utility functions for each bottom-level attribute and relative weights for all the attributes, in order to convert the natural scale values into utility scale values; and (e) computationally used a weighted, additive decision rule (or some other combination rule) to convert an alternative's scores on each of the bottom-level attributes into an overall utility score on the global MOE. Sensitivity analysis is routinely performed in MAUA to assess the impact of different scores, utility functions, and relative weights (or combination rules) on the overall MOE score for one or more alternatives. The big differences between cost-benefit analysis and MAUA is that the

former relies as much as possible on tangible (i.e., objective) benefits, and uses dollars instead of utilities as a metric for measuring value.

From a MAUA perspective, the omission of intangible benefits (and costs) is equivalent to omitting attributes from the MAU hierarchy. Whether this is acceptable or not depends on the nature of the "item(s)" being evaluated by a cost-benefit analysis. Lay (1985, p. 32) has discussed this point with consideration to expert systems. "Most capital investments decisions in the business field can be evaluated in terms of return on investment (ROI). This is because the asset that is being evaluated will create tangible benefits (such as the manufacture of a product for subsequent sale). An information system (particularly an expert system), may only produce intangible benefits [e.g., information and decision process support] and therefore the ROI criteria can no longer be applied. Intangibles, although not quantifiable, should be included in the process since their impact on the organization may be significant."

Obviously, we disagree with Lay's statement that intangibles are not quantifiable, for MAUA provides explicit procedures for quantifying the perceived value of intangibles. We do, however, agree with his focus on the significance of including intangibles in the evaluation. However, their inclusion or omission should depend on what factors the sponsoring and development teams consider to be important design and evaluation requirements. If intangibles are deemed unimportant enough to exclude them from the analysis, particularly after a thorough discussion of the advantages and disadvantages to including them, then it might be more appropriate to perform cost-benefit analysis than MAUA because of its greater familiarity and use as common business practice.

Actually, as Huber (1980, p.83) points out, the traditional cost-benefit analysis approach is a special case of the dollar-equivalent method, which is "... a special case of the more general ... MAU model technique." The appropriateness of the traditional cost-benefit analysis and dollar-equivalent methods versus MAUA depends on the defensibility of the conversions to monetary equivalents. If standard economic practices are clear and defensible, Huber argues that the traditional cost-benefit

analysis approach is often preferred because its conversions are more explicit and agreed-upon. However, as Huber (1980) and Riedel and Pitz (1986) point out, cost-benefit analysis requires substantial judgments that may be particularly subject to various biases As a result, "When conversion of the payoffs on all attributes to dollar equivalents seems reasonable and defensible, the dollar-equivalent technique is preferred over the MAU model technique. This is a consequence of the fact that the single aggregate figure derived in dollars can be more easily compared to the levels of other criteria that were not included in the analysis" (Huber, 1980, p. 82). As always, however, the needs and preferences of members of the sponsoring and development teams should be factored into the decision regarding which subjective evaluation method to use.

In closing this discussion, it is important to note that Keim and Janaro (1982) have argued for a phased cost-benefit analysis, where the nature of the analysis changes through the development cycle. Specifically, at the beginning of the effort, they argue for a relative cost-benefit analysis, where the focus is on identifying the relative costs and benefits of a range of alternative system configurations in order to select an alternative (or limited range of alternatives) for further specification. Their reasoning is that "... due to the evolutionary nature of the final system configuration the original estimates are often grossly distorted. The only way to make evaluations reasonable is to compare relative cost-benefit scenarios for the range of alternatives under consideration" (p. 25). As one moves through the different development steps, the system design becomes more specific; consequently, one can drop the relative analysis" focus because increasingly specific and quantifiable information is available for the system evaluation. Such a "phased" orientation is, of course, consistent with the discussion above of striving to link together requirements analysis (and the selection of alternatives) with their formal evaluations.

DECISION TREE ANALYSIS

Decision tree analysis is a formal method for combining uncertainties with utilities (or monetary equivalents) when evaluating alternative

decision options. There are numerous texts on the subject (e.g., see Brown et al., 1974; von Winterfeldt and Edwards, 1986; Watson and Buede, 1987). We will not discuss it in great detail because, at least to our knowledge, it has not yet been applied to evaluating expert systems. The interested reader is, however, referred to Cohen and Freeling (1981), who provide a detailed theoretical presentation of its potential applicability for evaluating information systems, and to O'Connor (1989), who discusses its applicability in developing and evaluating alternative architectures for the Strategic Defense Initiative.

What is particularly appealing about decision tree analysis for evaluating expert systems is the ability to use scenarios to represent, at a collective level, the uncertainties inherent in the decision-making situation facing members of the sponsoring team. Within decision tree analysis, these scenarios represent the members' hypotheses regarding alternative states of the world, a perspective that is consistent with the SHOR paradigm. Remember, at the broadest level, and particularly if the situation permits it during the earliest steps of the development process, the evaluator's job is to help members of the sponsoring team decide whether development of an expert system is an effective option for dealing with hypotheses regarding the current or future problem environment with which the organization will be dealing.

From a decision-analytic perspective, the overall utility or, more appropriately, expected utility, of different organizational options including whether or not to develop an expert system, depends on the (a) probabilities assigned to the various scenarios, and (b) the utility of each of the options for each of the scenarios. This situation can be illustrated by the concept of a payoff matrix, an example of which is presented in Table 3-1. The rows of the matrix represent all the different alternatives, including whether or not to develop the expert system, as well as variations on a particular theme, available to organizational decision makers (i.e., members of the sponsoring team). The columns represent the different scenarios that could significantly affect the attractiveness of the alternatives. The $p_1 \ldots p_k$ values represent the probabilities for each scenario, with their sum being 1.0. The cell

entries in the matrix indicate the utility (or value) of the outcome or "payoff" of each combination of options and scenarios. Each outcome is presumed to represent a cumulative payoff composed of perceived advantages and disadvantages on multiple criteria of varying importance to the decision maker(s). The "best" option is the one with the highest expected utility, which is calculated for each option by first multiplying the utilities for the outcomes and probabilities for the scenarios, and then summing the products.

Table 3-1: A Simple Payoff Matrix

	States of Nature			
Alternatives	$(p_1)S_1$	$(p_2)S_2$	...	$(p_k)S_k$
A	a_1	a_2	...	a_k
B	b_1	b_2	...	b_k
.	.		...	.
.	.	.	...	.
.	.	.	...	.
N	n_1	n_2	...	n_k

Substantial care must be given to defining the scenarios and obtaining the probability assessments. O'Connor and Edwards (1976) point out that not only do the scenarios have to be realistic, they have to be representative of a wide range of possible futures states of nature without being a long, tedious list of uncertainties. Moreover, they have to be capable of discriminating among the options in order to have any decision-making value. In short, they need to be an appropriate sample from the total scenario sample space.

With respect to probability assessments, "[t]he credibility of a scenario to a subject seems to depend more on the coherence with which its author has spun the tale than on its intrinsically 'logical' probability of occurrence" (Spetzler and Staël von Holstein, 1975, p. 347). Kahneman, Slovic, and Tversky (1982) have compiled an anthology of research studies demonstrating that, when compared to the tenets of probability and statistical theory, humans have limited appreciation for the concepts of randomness, statistical independence, sampling variability, data reliability, regression effects, etc. To quote Hogarth (1987, p. 40), "... we lead

statistical reasoning is entirely based on the logical structure of information, causal reasoning is responsive to both content and structure." Moreover, the causal implications of the stimuli can often mask the logical structure of the problem. Consequently, it is essential that the evaluator using decision analysis give substantial care to presenting the scenarios so that their logical probabilistic structure and, hence, relative likelihood can be better assessed by participating members of the sponsoring team. This often requires using a decision tree to decompose the scenario into the critical, uncertain events.

This point can be illustrated by considering an uncertainty dear to the heart of members of the development team, which is whether or not the sponsoring agency can provide the necessary funding level for the expert system throughout its development cycle. Figure 3-5 presents a highly simplified, hypothetical probability tree representing only two uncertain events: whether or not the funding environment is stable and, conditional upon it, whether or not the funding level will be satisfied.

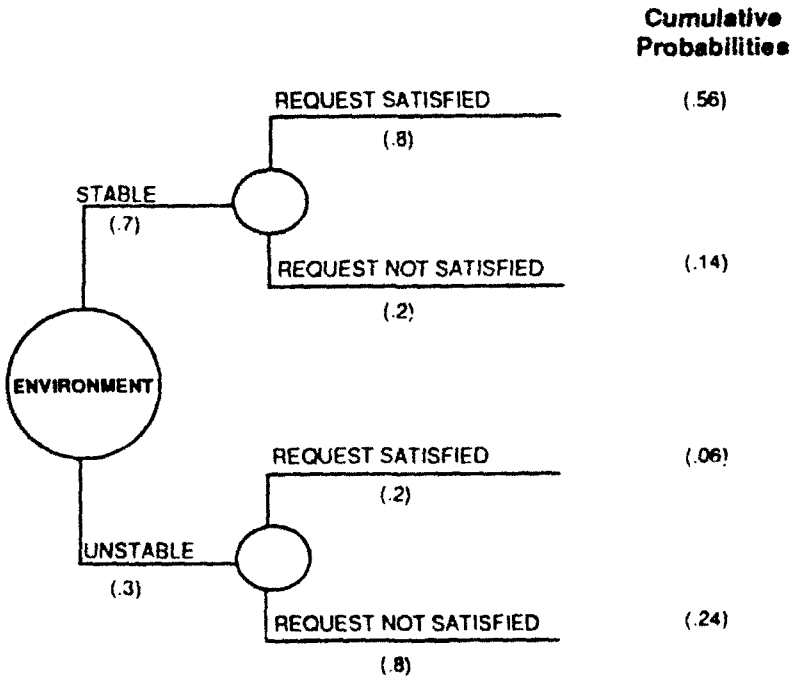


Figure 3-5: A Highly Simplified Probability Tree for Illustrating the Uncertainty in Funding for an Expert System throughout the Duration of the Development Process

As you can see, we are assuming a good state of affairs. A stable funding environment is considered twice as likely as an unstable environment. If the environment is stable, we are assuming that it is four times as likely as not that the development team will receive the necessary funding. If it is not stable, then we are assuming the opposite. If one multiplies out the probabilities for each branch of the tree and then sums the probabilities for the two branches resulting in the necessary funding for the expert system development effort, one finds, however, that the probability that the development team will have the necessary funding is actually only .62.

The situation gets somewhat more discouraging if one now considers the probability that the development team will develop an effective expert system that will be used by the decision maker(s) for whom it is being built. Figure 3-6 shows the probabilities for developing a "successful" expert system for each of the four branches of the tree in Figure 3-5.

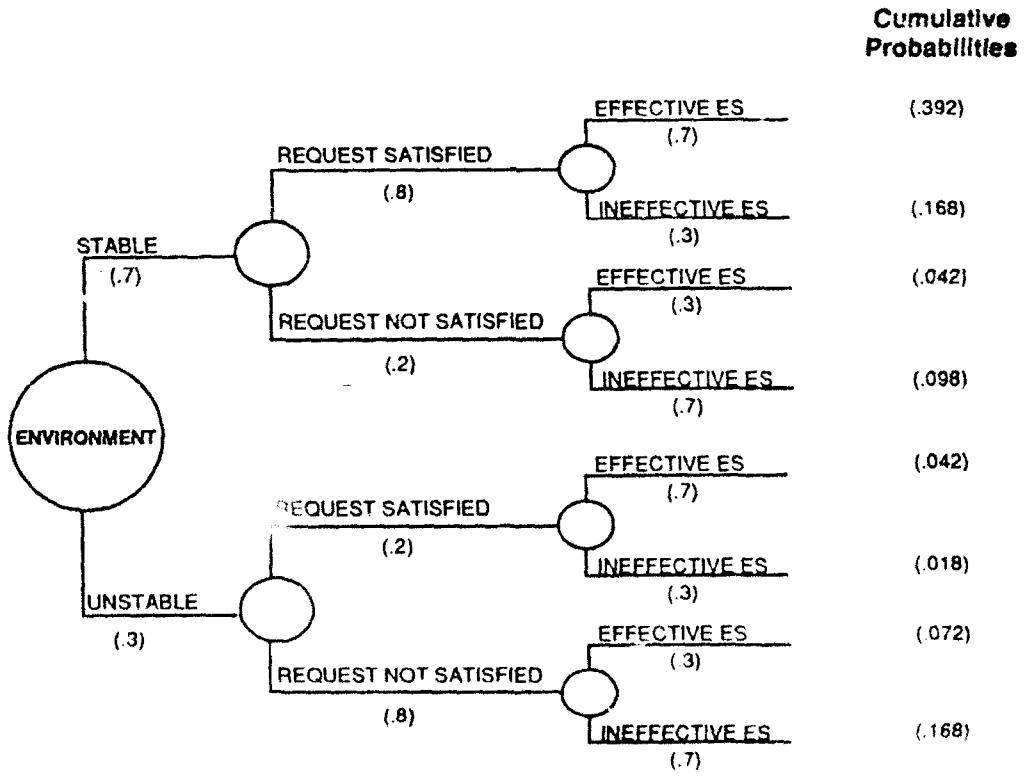


Figure 3-6: A Slightly Expanded Probability Tree for the Hypothetical Funding Decision

Again, we have assumed a good state of affairs--two-to-one odds for developing a "successful" expert system with the "necessary" funding. However, after multiplying-out all the probabilities in the tree and summing the probabilities for the appropriate branches, one finds that the probability of developing this "successful" expert system is 0.548—only a little better than flipping a coin.

The purpose in presenting what one might consider to be a reasonable, if not realistic, scenario is three-fold. The first purpose was to illustrate the importance of considering the structure of a scenario, not just its content. Substantial care must be given to eliciting probability assessments when using decision tree analysis, particularly the greater the ambiguity and the longer the time horizon for the uncertainties of interest, which is typically the case in the development process. The second purpose was to provide an alternative perspective on the sad fact that many expert systems are not successfully implemented. From a statistical perspective, a large number of things have to go right for successful implementation. And the third purpose was to again emphasize the importance of considering the uncertainties inherent in decision-making situations. As the example illustrates, it may be just as important for the development team as for the sponsoring team to consider these uncertainties. Decision analysis can alert members of the development team as to the uncertainties in the situation within which they will be working and, thereby, help further clarify the general requirements that the expert system will have to satisfy under various future conditions—for example, if all the "necessary funding" does not actually become available.

In closing this brief discussion of decision tree analysis, it is important to reiterate that decision tree analysis combines both probability and utility assessment. As was illustrated with the payoff matrix shown in Table 3-1, the "best" option is the one with the highest expected utility which is calculated for each option by first multiplying the overall values (i.e., utilities) for the outcomes and the probabilities for the scenarios, and then summing the products. The payoff matrix can be expanded (e.g., see Pitz and McKillip, 1984, p. 111) by using (a) a decision tree to pictorially represent scenarios and, thereby, reflect the

uncertainty in obtaining the outcomes for the options under consideration; and (b) a MAUA hierarchy to illustrate that the overall utility for an alternative, independent of the probability of obtaining it, is a composite score on multiple attributes. The expected utility for each option under consideration is the sum of the products for the probabilities for the scenarios and the utilities for the attributes. Thus, consistent with the SHOR paradigm, decision tree analysis is designed to assist decision makers in explicitly evaluating options in relation to hypotheses regarding the uncertainties inherent in the organization's future environment.

The process of performing a decision tree analysis is typically slow and difficult, however, for the decision-analytic representation of the problem can be quite large and the judgments quite extensive. Consequently, decision tree analysis is most viable if there is sufficient time (and resources) for the evaluator to work with the sponsoring team when it is still considering a range of options, that is, prior to Step 1 in the development cycle, and preliminary discussions suggest that uncertainties about the future environment may play a significant role in assessing the viability of developing an expert system. Once the development process is underway, however, the utility component is of most concern to test and evaluation since the probability of alternative future environmental states is not under the sponsoring or development team's control.

MAUA-BASED COST-BENEFIT ANALYSIS

MAUA-based cost-benefit analysis has been used to help design, completely on the basis of the decision makers' own judgments, the most beneficial option packages for various levels of dollar cost. Although this method is not as widely known as the subjective evaluation methods described above, it has been successfully used to develop advanced helicopter designs (Adelman, 1984), critical aspects of the U.S. Marine Corps' annual budget (Watson and Buede, 1987), health and hospital services (Weiss and Zwahlen, 1982), and the training curriculum for a federal government agency (Medlin and Adelman, 1989).

The MAUA-based cost-benefit analysis approach has six basic steps: (1) divide the problem into independent areas (or "variables") over which benefits and costs can vary almost independently; then, (2) identify distinctly different actions (or "levels") on each variable that increase in benefit and cost; (3) assess the relative benefit and cost of each level on each variable; (4) assess the relative benefit of one variable against another by using relative weights on the variables; (5) calculate the change in benefit to the change in cost ratio for each level of each variable as one moves from the lowest to the highest level of each variable; and (6) use an optimization algorithm to calculate the efficient allocations defining the most beneficial package (i.e., one level on each variable) for varying degrees of (total) cost.

When selecting a set of expert systems, the different variables represent the different areas for which expert systems are being considered by the sponsoring team. For example, assume that an organization is considering the development of expert systems for each of three major divisions (A, B, and C) within the organization. The initial level on a variable (e.g., A) may represent either the status quo, which may be "no expert system," or the cheapest, most "bare-bones" concept for developing that expert system. In the case study, the status quo of "no expert system" is represented by level #0; the "bare-bones" concept is represented by level #1. The last level on a variable represents the most expensive, "gold-plated" (yet realistic) conceptualization of the expert system for that area. The intermediate levels on a variable represent intermediate conceptualizations of the expert system as one moves from the "bare-bones" to more "gold-plated" concepts.

A relative benefit scale is established for each variable such that the initial level is given a value of zero and the "gold-plated" concept is given a value of 100. Paired comparison techniques are typically used to determine the relative benefit of the intermediate conceptualizations of the expert system on the variable. In particular, the focus is on how much benefit an intermediate level provides between the two endpoints of the variable's scale—that is, between the "bare-bones" and "gold-plated" concepts. For example, is conceptualization #2 (e.g., level #2 on

variable A) halfway in benefit between the "bare-bones" and "gold-plated" concepts? If the answer is "yes," then conceptualization #2 would get a benefit score of 50 on that variable. If the answer is "no," then the questioning focuses on how much less than 50. For example, is the relative benefit 10% of the way between the "bare-bones" and "gold-plated" concepts or 25% or 40%, etc.? Once a relative benefit score is obtained, the focus shifts to conceptualization #3 on the variable, which must have a relative benefit score between that for conceptualization #2 and the gold-plated conceptualization. In answering these and related, relative-value questions, a subjective benefit scale is developed for each level of each variable. [Note: A MAUA hierarchy with utility functions and weights can be used too, although it is obviously a more complex approach than obtaining paired-comparison benefit values.]

Relative importance weights are then used to indicate the relative benefit of improving (from the initial level to "gold-plated") on each variable (i.e., A vs. B vs. C in our example). For example, let's assume that going from the status quo of "no expert system" to "gold-plated on variable A was thought to be twice as beneficial as doing so on both variables B and C, which are equally important. If the relative importance weights sum to 100, then the relative weight on variable A would be 50 and the relative weights on variables B and C would be 25. The overall benefit given to any level on any variable can be compared to that for any other level of another variable (except the initial level for a variable, which is set to zero to indicate it's the starting point) by multiplying (a) the relative weights for the variable and (b) the benefit value for the level within the variable. For example, let's assume that conceptualization #3 for variable A had a within-variable benefit value of 50. Since, in our example, variable A has a relative weight of 50 and variable B has a relative weight of 25, conceptualization #3 on variable A has the same overall benefit as the gold-plated concept on variable B because 50 × 50 equals 100 × 25. [Note: The overall benefit of each level for each variable in the design could be assessed directly using paired comparison techniques. However, experience suggests that participants find the above procedure easier, particularly when there are many levels and variables.]

In addition, a cost estimate also needs to be obtained for each conceptualization, that is, for each level of each variable. The cost estimate for the first level on each variable is important only in providing a reference point, for the analysis assumes that the starting point (or first package of expert systems) is represented by the initial level on each variable. The cost estimates could be in absolute dollars or in "relative costs," depending upon which one the participants (or more likely, the personnel responsible for software cost estimation) feel more confident in using in the analysis. [Note: Conceptually, the cost could be any resource with allocation constraints.] At this point one can calculate the overall incremental benefits and costs of moving from one level to another on each variable.

The goal is to maximize the benefit for the set of expert systems at any given level of total dollar costs—that is, to define the "efficient frontier," such as the hypothetical one shown in Figure 3-7. Starting with the first level on all the variables, which we have defined as the set of expert systems with the lowest possible benefit relative to any other set, we will follow three steps. First, we will calculate the incremental change in benefit to change in cost ratio for each level of each variable as one moves from the lowest to the highest level of each variable. Second, we will order the levels on the basis of this ratio. Third, we will sequentially select the level with the highest change in benefit to change in cost ratio. Thus, each incremental point on the efficient frontier will represent a set of expert systems that was identical to the one that preceded it except for one change, that remaining level with the highest change in benefit to change in cost level at that time. [Notes: If the analysis shows dips in the level-to-level analysis, incremental benefit-to-cost ratios are calculated across multiple levels. Although this algorithm may not derive all the points on the efficient frontier, all the points derived are on the frontier. Moreover, experience has shown that it is easier than other approaches (such as integer programming) for decision makers to understand the algorithm and follow its implications. Finally, it can be readily programmed for, and will operate quickly on, personal computers.]

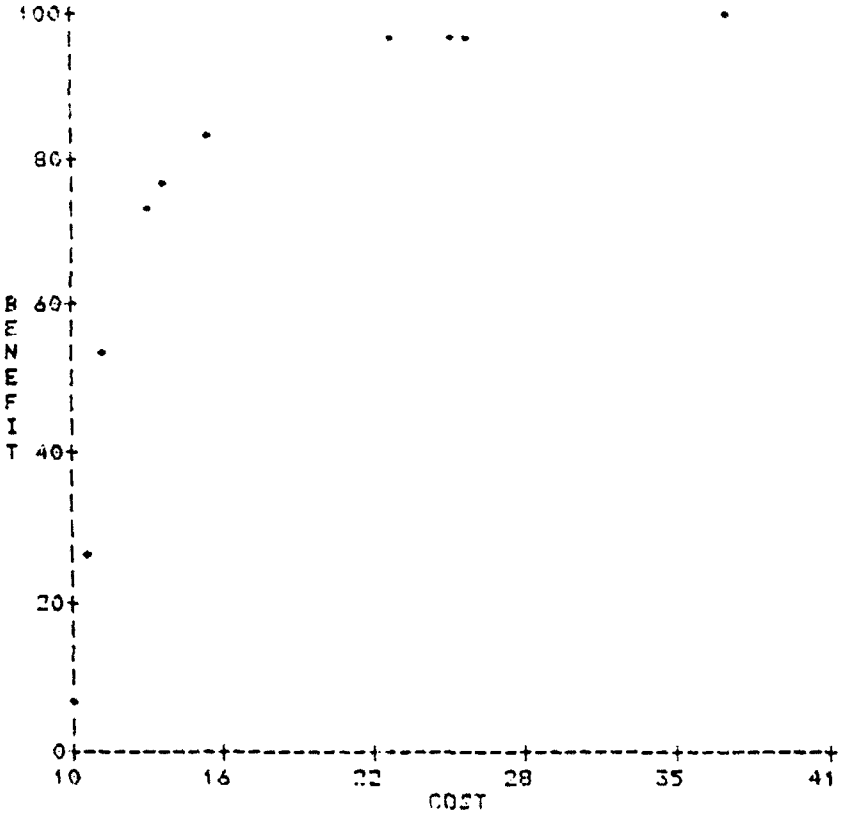


Figure 3-7: A Hypothetical Efficient Frontier

The MAUA-based cost-benefit analysis can also be used to identify the most beneficial configuration of components of a particular expert system at different levels of cost. In this case, the different components of the expert system represent the different variables. The different potential levels of sophistication of each component represent the levels on the variables. Relative benefit values are obtained within and between variables, and costs are obtained for each level. And the algorithm described above is used to generate points on the efficient frontier indicating the package of component parts providing the most benefit at different levels of total cost.

CONSTRUCTING QUESTIONNAIRES TO ELICIT OPINIONS

Thus far, Chapter 3 has considered a number of different subjective test and evaluation methods, most notably MAUA. The focus on these methods has been toward helping the sponsoring and development team: (a) identify

evaluation criteria early in (if not prior to) development to guide the development (as well as testing) process; (b) convert test scores into utility measures; and (c) utilize explicit procedures for weighting, or in some other fashion integrating test results on all the criteria into an overall assessment of the expert system's adequacy. As important as obtaining an overall score for the expert system is assessing the expert system's weaknesses, particularly for important criteria. This feedback can, in turn, guide subsequent development efforts and, thereby, effectively integrate test and evaluation into the development process.

In this section of Chapter 3, we consider the construction of subjective questionnaires for obtaining potential users' opinions about the expert system. In particular, we are concerned about users' judgments of the expert system's performance and usability. Questionnaires were used to obtain these judgments for the DART expert system described in Chapter 1. In this section, we want to go over the basic issues inherent in questionnaire construction. Throughout the discussion we will assume that the users' responses to the questionnaire are used to score the expert system on subjective criteria in the evaluation hierarchy, such as in the one described in the next section of this chapter (and displayed in Table 2-1).

We begin this section by first defining two critical measurement concepts, reliability and validity. Reliability means that the measurement instrument (e.g., questionnaire) gives the same results when it is used on two different occasions. The key idea here is "replication;" one can repeat the measurement process with the same result. A basic assumption is that there have been no changes in the object being measured (e.g., the expert system) in-between the two measurement periods. By "validity" we mean that the instrument is measuring what it is supposed to measure. An instrument can be reliable (i.e., it produces the same results upon replication), but invalid (i.e., it reliably measures the wrong thing). However, an instrument cannot be valid if it is totally unreliable because the latter implies that it will give very different answers when used to measure the same thing on two or more occasions.

We now consider how these two concepts were assessed for the questionnaire used in the test and evaluation of the DART expert system. (For more information, see Adelman et al., 1985.) Prior to doing so, however, we will review the characteristics of the DART questionnaire. These are the characteristics you should have in your questionnaires, assuming, of course, that you use the same kind of questionnaire to obtain users' opinions about an expert system. It is important to note that there are other types of questionnaires. Two other types will also be overviewed later in this section.

Characteristics of the DART Questionnaire

As you remember, the DART questionnaire was designed to obtain users' opinions about DART with respect to the evaluation criteria identified in Table 1-1. The questionnaire had a total of 121 questions. Most of the questions assessed the bottom-level attributes in Table 1-1. However, 6 questions directly assessed overall utility (node 0.0 in Table 1-1), 2 questions directly assessed decision process quality (node 3.3 in Table 1-1), and 3 questions each assessed the quality of the training sessions and test scenarios (neither of which were evaluation criteria in Table 1-1, although important to the test and evaluation team to assess for more general reasons).

Two questions from the DART questionnaire are presented below so that one can get a feeling for the kinds of questions used in the questionnaire. The first question measures "response time" (attribute #1.2.1.4 in the hierarchy); the second question measures "acceptability of time for task accomplishment" (attribute #2.1.1.1).

I had to wait too long for the DART aid to respond to my inputs.

Very Strongly Disagree				Neither Disagree Nor Agree					Very Strongly Agree	
0	1	2	3	4	5	6	7	8	9	10

Use of the DART aid will not slow down the identification process now used in the Tactical Air Control System.

Very Strongly Disagree				Neither Disagree Nor Agree					Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

As can be seen, all questions required the participant to respond on a eleven-point scale from 0 (very strongly disagree) to 10 (very strongly agree), with 5 being "neither disagree nor agree." This type of scale is referred to as a Likert (1932) scale after Rensis Likert, the psychologist who first developed it. The length of the scale (i.e., eleven points) and the end points (i.e., 0 and 10) are arbitrary. We could have used a 3-, 5-, 7- or whatever point scale we wanted. We chose an eleven-point scale in order to give the users plenty of room to express the extent to which they agreed or disagreed with each question which was written in the form of a statement. The use of only positive numbers for the scale values was also arbitrary. We could have used negative numbers to represent disagreement and the 0-point to represent "Neither Disagree Nor Agree." We chose to use positive numbers because, as was illustrated by the first question above, sometimes we wanted the user to disagree with the statement in order to score DART highly. Therefore, we were concerned that the use of negative numbers might be confusing.

There were two or more questions for each MOE criterion in an effort to achieve greater confidence in the criterion scores. In addition, this permitted us to calculate a split-half reliability measure, which is described in the next subsection. The number in the parentheses to the right of each bottom-level attribute in Table 1-1 indicates the number of questions assessing it. The actual number depended on the availability of previously written questions assessing the criterion (e.g., from Sage and White, 1980), the ease in writing "different-sounding" questions for the criterion, and its depth in the hierarchy. We tended to use more questions when we were measuring bottom-level attributes high in the hierarchy. For example, we used seven questions to measure "decision accuracy" (attribute #3.1), but only two questions to measure "response time" (attribute #1.2.1.4).

Half the questions for each criterion were presented in each half of the questionnaire in an effort to prevent the questions' order in the questionnaire from affecting the attributes' scores. And, as will be seen, this procedure is also appropriate for calculating a split-half reliability measure. In most cases, a high score indicated good performance, but typically for one question measuring each criterion, a low score indicated good performance in an effort to ensure that the participants paid careful attention to the questions. Prior to calculating attribute scores, the users' responses were rescaled as if the question were asked in a positive fashion. DART's score on a bottom-level attribute was the mean score of the participants' responses to the questions assessing it. Values for criteria moving up the hierarchy were the mean score for the criteria below it.

As we noted in Chapter 1, by averaging lower-level attribute scores to obtain upper-level criterion scores, one is giving each criterion equal weight at its place in the hierarchy. For example, by averaging the mean scores for "training" (attribute #1.1.1), "work style" (attribute #1.1.2), and "operational needs" (attribute #1.1.3), each of three attributes received a relative weight of 0.333 in determining the score on "match with personnel" (attribute #1.1). Although it was quite possible that the participating domain experts may have thought that certain bottom-level criteria were more important than others, members of the test and evaluation team thought it inappropriate to have the (DART) experts differentially weight these criteria at the time of the evaluation because we wanted to use the same weights for evaluating all five prototypes being developed on the contract in order to provide a common evaluation baseline.

In closing this subsection, we want to emphasize that you should keep the following points in mind when developing the questions for your questionnaire. First, remember that people do not like completing questionnaires. Some people complete them as quickly as possible, often not reading the questions carefully. Other people seem to scrutinize every word and nuance in the question, just trying to find something wrong with it. Consequently, try to keep the questions short and to the point. Do not use qualifying phrases in a question if you can help it because

respondents may inadvertently respond to the qualifying phrase instead of the principal one. In a similar vein, minimize the use of the word "not" because respondents sometimes misinterpret it or fail to recognize it when they are rushing through a questionnaire.

Second, have a colleague critically review your questions. Ask that colleague to suggest better ways of asking any questions they are having trouble answering. Third, pilot-test your questionnaire with representative users before you actually use it to obtain users' opinions of an expert system. Ask the respondents to think aloud when they answer the questions so you can assess whether others are interpreting the questions in the way that you intended. If they have no objections, tape-record the session so that you don't have to rely on your memory. Revise questions that are being misinterpreted by the pilot participants during the session to see if you can reword them in a way that removes the ambiguities. Continue pilot-testing the questionnaire until most (if not all) questions are interpreted in the way you intended.

Reliability and Validity of the DART Questionnaire

We now turn to consider how we assessed the reliability and validity of the DART questionnaire. Since only four technical representatives and three substantive domain experts participated in the DART test and evaluation, the reliability and validity assessments used the responses from all the technical representatives and domain users who participated in testing and evaluating the five decision-aiding system prototypes developed on the contract. Remember, there were two evaluation sessions for each system. In all cases, technical representatives from the Rome Air Development Center (RADC) participated in the first session, and Air Force substantive experts in the decision task for which the system was designed participated in the second session. In general, each session followed the same format: the first day was dedicated to providing a detailed overview of the system; the second day was dedicated to providing the participants with "hands-on" training in using the system; on the third day the participants worked test problems with and without the system; and on the fourth day the partici-

pants completed the questionnaires and discussed the system prototype with members of the test, development, and sponsoring teams.

In total, 15 Air Force substantive experts and 13 RADC technical representatives participated in the sessions. The substantive experts, all of whom were selected by the Tactical Air Command, had years of experience in the tactical decision-making area for which the system was developed; most had minimal computer science training. In contrast, the technical representatives had minimal, if any, substantive expertise in the areas for which the aids were developed. Eleven of the technical representatives were Air Force personnel who, in most cases, had just recently received an undergraduate degree and taken computer science coursework; the other two technical representatives were civilians with technical backgrounds who had worked on RADC projects for at least two years. In a number of cases, the same technical representative participated in more than one evaluation. In order to ensure that the results presented below were not skewed by the opinion of these participants, we used only the questionnaire responses from their first evaluation session. Finally, it should be mentioned here that the technical representatives' responses for one of the five systems were not included in the analyses because the system was not functioning sufficiently well to permit an accurate assessment of its strengths and weaknesses.

Assessing the Reliability of the DART Questionnaire

Split-half reliability and test-retest reliability measures were calculated. Split-half reliability is a measure that relates the two halves of the questionnaire. A split-half reliability measure was possible because two or more questions were used to assess the participants' responses for each attribute in the MAUA evaluation hierarchy, and the questions were divided between the two halves of the questionnaire. If the questionnaire was a reliable measurement instrument, then there should be a high correlation between the two halves of the questionnaire, for presumably the questions were measuring the same attribute. The following formula from Gulliksen (1950) was used to calculate the split-half reliability of the questionnaire:

$$r''_{xx} = 2\left[1 - \frac{s^2_{x_1} + s^2_{x_2}}{s^2_x}\right], \qquad [3\text{-}2]$$

where

r''_{xx} is the split-half reliability of the questionnaire,

$s^2_{x_1}$ is the variance of the first half of the questionnaire,

$s^2_{x_2}$ is the variance of the second half of the questionnaire, and

s^2_x is the variance of the sum of the scores on the two halves of the questionnaire ($x = x_1 + x_2$).

The split-half reliability measure was 0.741 for the substantive experts and 0.707 for the technical representatives; both reliability measures were significantly different than zero at the $p < 0.01$ level ($df = x_1 = 58 - 2 = 56$).

The most conservative measure of a questionnaire's reliability is obtained by re-administering the questionnaire a second time after a month or more has passed, and then correlating the participants' responses to the questions. Three of the technical representatives, each for different prototypes, agreed to complete the questionnaire a second time. Six to eight weeks separated the second completion of the questionnaire, in an effort to ensure that the participants remembered the prototypes' general characteristics but not their responses to specific questions. The three test-retest correlations were 0.44, 0.61, and 0.56. Although these correlations may seem low to the reader, it must be remembered that, unlike questionnaires assessing personality or attitude traits which are presumed to be stable and unchanging, we were assessing the participants' memory of the system's many characteristics, which is presumed to be more unstable and subject to change. All three correlations were significantly different from zero at the $p < 0.01$ significance level ($df = x_1 + x_2 - 2 = 114$).

These results indicate that the questionnaire is a reliable instrument, that both halves are reliable, and that if the questionnaire is used to measure an expert system at two different times and there is no difference in the system in the interim, the tester will obtain the same opinions from test participants.

Assessing the Validity of the DART Questionnaire

Three different forms of validity are important for questionnaires. First, face (or content) validity implies that, at least on the surface, a questionnaire appears to be measuring what it is supposed to be measuring. Face validity was assured in our questionnaire by having a retired Air Force lieutenant colonel—who was a substantive expert in the tactical decision-making problems for which the prototypes were developed—write the questions.

Second, predictive (or external) validity implies that the instrument is consistent and agrees with another established measure of the same attribute. To measure the questionnaire's predictive validity, we used the same basic approach as that used by Bailey and Pearson (1983); we correlated the participants' global evaluations of the prototype with the results of the questionnaire. Specifically, we correlated the participants' mean responses to the six questions directly asking about the prototype's overall utility with the participants' scores for the prototypes based on the MAUA evaluation hierarchy (i.e., the value for node 0.0). The correlation for the 15 experts was 0.85 ($p < 0.01$, $df = 13$), and the correlation for the 13 technical representatives was 0.60 ($p < 0.05$, $df = 11$).

Third, construct validity examines the theoretical adequacy of the components of the construct being measured, typically by comparing the scores obtained from two separate measuring instruments aimed at the same construct. We reasoned that if the questionnaire had construct validity, then it should be possible to relate aspects of the system prototypes that the participants indicated they liked and disliked in the open-ended questionnaire to the attributes in the MAUA hierarchy that were scored high

and low, respectively. Two specific steps were required to calculate a measure of construct validity. First, for each prototype, we matched those aspects of the prototype that the substantive experts indicated they liked or disliked in the open-ended questionnaire to specific attributes in the MAUA hierarchy. Second, for each prototype, we rank-ordered the attributes according to their mean score on the eleven-point scale. We found that 78 percent of the matched attributes fell into either the top 30 percent or the bottom 30 percent of the distribution of rank-ordered attributes, thereby indicating a relationship between both of our questionnaires and construct validity.

Other Types of Questionnaires

In this last subsection, we will consider two other types of questionnaires. The first type is the traditional open-ended questionnaire; the second type is designed to directly score the system on the bottom-level attributes in a MAUA hierarchy. Although we will not provide additional analysis here, the reader should remember that reliability and validity are also important concepts for these questionnaires too.

An open-ended questionnaire is analogous to an interview in that it gives respondents an opportunity to say what they want to say. In fact, an open-ended questionnaire should be given in conjunction with an interview or round-table discussion. The use of open-ended questionnaires, interviews, and discussions is important because we don't want to loose critical information simply because we didn't ask the right question. More importantly, we want to give the users an opportunity to elaborate on their numeric answers. For example, we want them to tell us why they gave the system a "0" on "usability" or "confidence" or whatever. Moreover, we want them to tell us what changes, in their opinion, would improve the system's scores on those criteria for which it is scoring poorly. Although the numbers tell us how the expert system is scoring in the users' eyes, it doesn't tell us why this is so. To obtain the latter, one needs to use some form of open-ended questionnaire or interview.

The ten questions used in the DART open-ended questionnaire are presented below. We provided space after each question for the users' responses, although it is omitted below. In examining the questions you will notice that, while we wanted the users to tell us (a) what they considered the system's strengths and weaknesses to be, and (b) how to improve the system, we still gave them some direction in order to focus their responses. In addition, we used the questionnaire as a means to document the users' experience and background.

1. What did you like and/or find most useful about the DART aid?

2. What did you dislike and/or find to be a hindrance about the DART aid?

3. What specific changes and/or modifications would you suggest regarding the following characteristics of the DART aid? Write NONE if you have no suggestions for improving the particular characteristics:

   a) regarding the aid's general technical approach to identification, and its degrees of belief;

   b) regarding the different types of expert knowledge stored in the aid;

   c) regarding the value of the explanation mechanism in the DART aid;

   d) regarding the user interface with the DART aid;

   e) regarding the DART's graphic displays.

4. What would you envision to be the future potential of this aid in your present or most recent operational environment. Why?

5. Were the instructions sufficient to enable you to efficiently use this aid? Comments?

6. Where do you feel a second-generation, operational version of the aid would receive good acceptance. Why?

7. What would you envision to be the future training potential of the DART aid? Why?

8. Please state your experience in performing the task you performed here today. Also, please state relevant duty assignments, and the number of years you performed them.

9. Please state your level of experience and training with computers and decision aids.

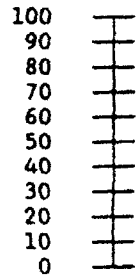
10. Please give any other comments that you feel are relevant to this questionnaire.

The second type of questionnaire is designed to generate utility scores for the expert system on the bottom-level attributes in an MAUA evaluation hierarchy; that is, the user scores the system on a 0 to 100 (or 0 to 1.0 or 10 or whatever) value scale for each of the bottom-level attributes. For example, instead of answering a series of questions about the system's response time, the user would score the acceptability of the system's response time on a 0 to 100 acceptability scale.

It is important to note at the outset that this type of questionnaire is difficult for users to complete, for it represents the utility scoring part of MAU analysis but without access to an analyst or computer program. By "utility scoring" we mean that it not only involves scoring the expert system on a subjective scale (e.g., for "ease of use"), but then translating that performance score into a utility score (e.g., the acceptability of the system's "ease of use"). In the approach described below, acceptability is defined relative to the extent that the system meets the user's performance expectations for a criterion. Here we will briefly introduce some important concepts to keep in mind when developing such a questionnaire.

Figure 3-8 presents an example of a questionnaire for obtaining utility scores on the bottom-level attributes measuring "Performance:Judgment" and "Usability" in Table 2-1. In particular, Figure 3-8 is requesting that the user indicate the adequacy of the system's "Response Time Performance" in terms of the amount of time the expert system took to respond to the operator's inputs and provide outputs. The "50" point means that the system fully meets the user's performance expectations for the system on the evaluation criterion being considered (in keeping with the scale recommended in the last section of this chapter). The "0" means the system fails the performance expectations. The "100" means the system not only fully meets the performance expectations, but greatly exceeds them.

1. RESPONSE TIME PERFORMANCE



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no.")

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

  _____
  _____
  _____
  _____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

  _____
  _____
  _____
  _____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

  _____
  _____
  _____
  _____

Figure 3-8: An Example of a Questionnaire for Obtaining Utility Scores

More generally, scores below "50" mean that, in the user's judgment, the system was in some fashion deficient on the criterion; scores above "50" mean that the system was providing added value on the criterion. The scale permits the user to numerically score the level of deficiency or the level of added value. For example, let's consider evaluating the system on Response Time. If the system met the user's performance expectations for an acceptable waiting period between the inputs and the system's response to them, then the user would give it a score of "50." Let's assume that the user considered the system's response time deficient (i.e., less than "50"), but not a complete failure (i.e., greater than "0"). Then the question is, "What is its numerical level of deficiency between 0 and 50?" If the deficiency was very minor in the user's mind, then the score would be close to 50 (e.g., greater than or equal to 45, but less than 50). On the other hand, if the deficiency was very great but still not "0," then the score would be close to 0 (e.g., less than or equal to 5, but greater than 0). If the user thought the level of deficiency was about halfway between meeting the expectation and failing it, the user would give the system a score of 25; if it was a quarter-of-the-way, he or she would score it 12.5 and so forth. In short, the user would use the bottom-half of the scale to numerically specify the expert system's level of deficiency on the evaluation criterion. In addition, of course, testers need to know the reason(s) for the user's score. Consequently, space is provided to tell us what the user's performance expectations were, assuming he or she had ones, and the reasons for the score on the criterion.

In a similar fashion, the user can use the scale between 50 and 100 to numerically specify the level of "added value" performance on the criterion. For example, if the system barely exceeded the user's performance expectations for Response Time, then it would receive a score slightly above 50. If it considerably exceeded the performance expectations but was not a 100, it might receive an 85, 90, 95, etc. If the degree of added value benefit provided by the system was about halfway between meeting the user's performance expectations and greatly exceeding it, then you would score it 75. If the added-value benefit was a quarter-of-the-way, the system would receive a score of 62.5; if it was three-quarters, it would receive a score of 87.5 and so forth. Again, it is important to know the

system is providing added-value on the criterion—that is, the reasons for the score.

The questionnaire would contain such a scoring sheet, or some derivative of the above approach, for each bottom-level attribute in the MAUA evaluation hierarchy requiring the user's opinion of the expert system. As you can imagine, these are not easy judgments to make. As a means of helping users, we first ask them to think about their performance expectations for the system on the criterion. What level of performance do they consider acceptable (i.e., a score of "50")? We give them room to write their performance expectations on the scoring sheet. We also give them an opportunity to indicate (a) whether they have previously expressed performance expectations for the criterion and (b) whether they have heard anyone else express performance expectations. Then we ask that they provide a numerical score, and the reasons for it, in the space provided. If they cannot (or do not want to) score the system on a particular criterion, we do not force them to do so. A number of omissions would indicate the inadequacy of this type of questionnaire, and we would subsequently use the Likert-type with open-ended questionnaires and/or interviews.

Summary

All the questionnaires we've considered here attempt to capture the users' opinions about the expert system. Open-ended questionnaires, interviews, and round-table discussions are important because they give users an opportunity to indicate what they liked and disliked about the system, and how they would improve it. The short-answer, Likert-type questionnaires are important because they attempt to quantify the users' opinions. In particular, by building the short-answer questionnaire around an MAUA evaluation hierarchy, this familiar type of questionnaire provides a means for scoring the expert system on the more "subjective" attributes in the hierarchy. Finally, the "utility" questionnaire attempts to go one step further and translate the scores on these attributes into utility values.

With the Likert-type questionnaire used in the DART test and evaluation, we assumed a linear scale for converting the users' responses (i.e.,

"scores") into utility values on the attributes. The utility questionnaire does not make this assumption but, instead, attempts to directly assess the utility values using the "50" point as a reference point. For example, a user might give an expert system a score of "7" (using the 0 to 10 point Likert scale) on "response time." However that "7" may or may not actually meet the user's expectations for that criterion. By attempting to measure the extent to which the system meets the user's expectations for "response time," we are attempting to assess the value that the user places on the expert system's performance on this attribute.

In closing this subsection, it is important to emphasize that the type of responses being elicited with a "utility" questionnaire, although difficult, are at the heart of quantifying the value that users place on the expert system's performance for all the attributes, not just the more subjective ones. Remember, "scoring" and "weighting" are two separate steps in applying MAUA. The utility scale converts the expert system's scores on the different scales being used to measure the different attributes (i.e., the proverbial "apples and oranges") into a common value scale. The relative weights indicate the attributes' relative importance. How far one wants to go in the process of converting test scores into overall utility scores is a critical question for the test and evaluation team to consider. It is our opinion that one needs to implement the entire MAUA process to most effectively focus expert system development on the users' objectives and, thereby, integrate test and evaluation into the development process.

The Appendix of this volume contains both a Likert-scale questionnaire and a utility-scale questionnaire that can be used in conjunction with the MAUA framework presented below to test an expert system. These questionnaires are complete with instructions on their use, and they have been designed according to the guidelines described above.

PROPOSED MAUA FRAMEWORK FOR TESTING AND EVALUATING EXPERT SYSTEMS

The rest of this chapter contains the description of our proposed MAUA framework for testing and evaluating expert systems. The attributes in

this framework were developed iteratively by top-down decomposition of important aspects of expert systems, and bottom-up aggregation of software quality metrics. The specific attributes included were identified as a result of our own research and a review of related work by Ulvila et al. (1987), Riedel and Pitz (1986), Rockmore et al. (1982), Kirk and Murray (1988), Adelman et al. (1985), Klein and Brezovic (1988), and Tong et al. (1987). The attributes described are generic and are potentially applicable to any expert system. However, the relative importance of an attribute will be determined by specific features of the system and its intended use. We discuss this more in Chapter 7. Similarly, the measurement scales for the attributes may vary from one system to another, and the proper measurement technique (subjective, empirical, or technical) will vary from attribute to attribute and from system to system depending on the nature of the test, the resources available for the test, the importance of the attribute, and other conditions. We do, however, provide guidance for developing measurement scales, and we present, in the Appendix, two questionnaires that can be used to assess the system against judgmentally determined performance attributes and most usability attributes.

In addition to being generic, the framework is also comprehensive. It purports to address all important aspects of expert systems. In this regard, it differs from most of the expert system verification, validation, and testing work done to date, including that referenced above. Some researchers are paying attention to assessments of an expert system's knowledge-base structure and content. Others stress the quality of the system's answers. None of the other work addresses both of these aspects and ties them together with service requirements and individual and organizational usability. Still, specifics of an expert system or its intended operation may require a tester to add a few unique attributes.

An experienced software tester will notice that one aspect of conventional software testing is left out of our framework—namely, software design and coding standards of the type address by DoD-STD 2167, DoD-STD-1679A, MIL-STD-1679, and JCMPOINST 8020.1 for conventional software. Ulvila et al. (1987) attempt an application of these standards to Common Lisp, a widely used expert system language. The fit is not especially

good, and we feel that the state of development of generally accepted software engineering practice for expert systems is not yet developed enough to have a codified set of good or acceptable design and coding standards. Consequently, these items are not in our set of attributes. As the field of expert systems matures, such standards may be developed and they should be added to our framework at that time.

Chapter 7 demonstrates how this framework is used to pull the testing effort together. A key aspect in "pulling it all together" is a four-step approach to using the framework in Figure 7-1 (which is a reproduction of Figure 3-9). First, establish the relative importance of the different major areas (the top level in the framework: knowledge base, inference engine, "service," performance, and usability), then sub-areas, and then attributes. This information is then refined into weights. Second, examine each attribute, determine its measure, and determine how to collect that information. Third, collect that information about the system being tested. Fourth, process the information through the MAUA. Fifth, evaluate the results by comparisons between actual results and the desired or required results.

There are other views of software testing. These include: software quality metrics, validation versus verification, and static versus dynamic testing. The framework that we present is compatible with these other views of testing as discussed in Chapter 8.

Framework and Attribute Definitions

Figure 3-9 shows our MAU proposed framework for testing and evaluating expert systems. The overall assessment of the expert system is composed of five criteria: knowledge base, inference engine, service requirements, performance, and usability. These are subdivided to the level of attributes as described below.

KNOWLEDGE BASE. These attributes refer to the structure and content of the expert system's knowledge base. While the descriptions below are phrased in terms of a rule base, analogous attributes would apply to a

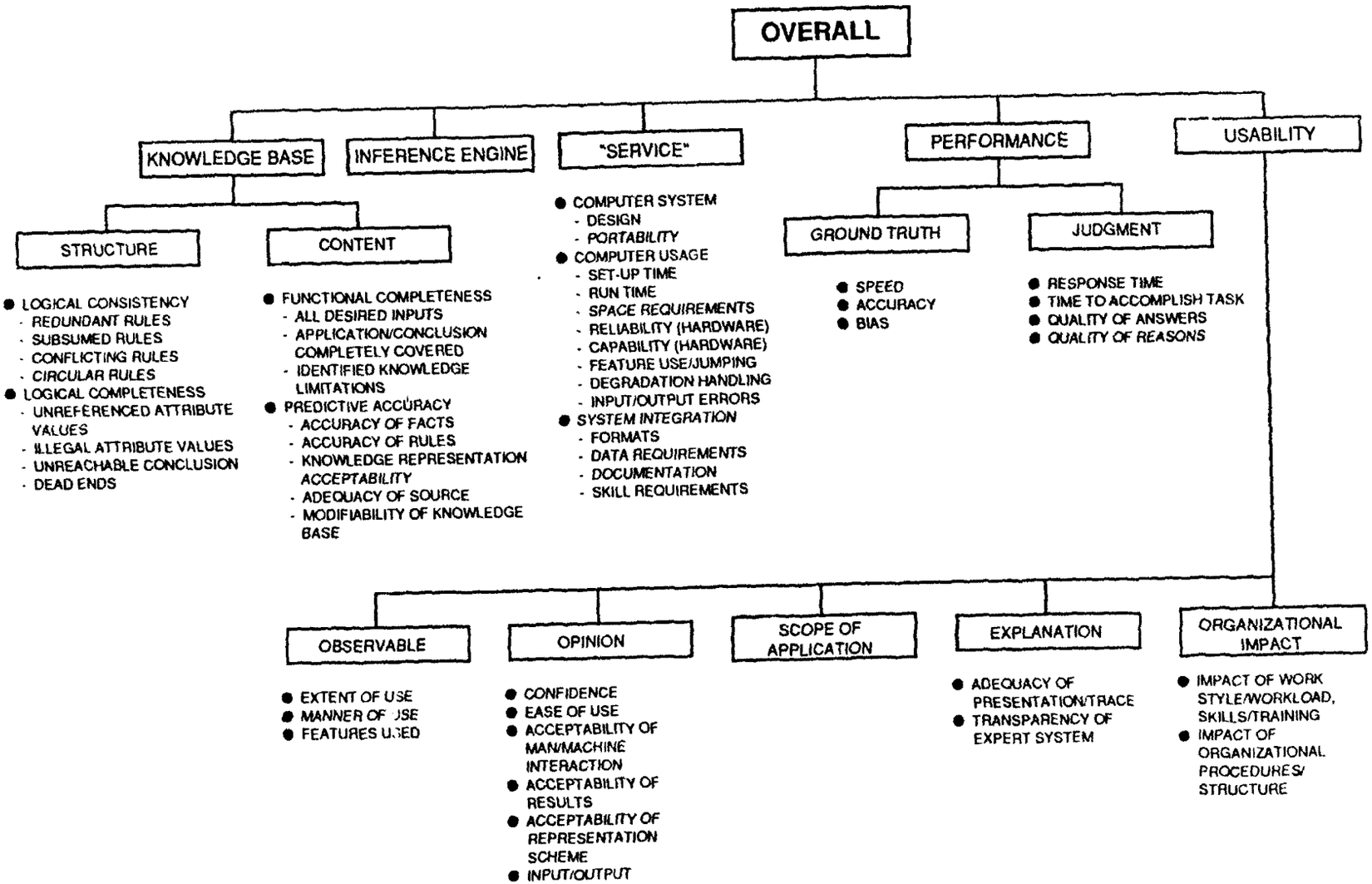


Figure 3-9: A MAU Framework for Testing and Evaluating Expert Systems

frame-based expert system. (See Hayes, 1981, for a discussion of the logical equivalents of rule-based and frame-based systems.)

Structure

Logical Consistency. The following attributes would limit the consistency (or correspondence) and efficiency of a knowledge base. Redundant rules are rules or groups of rules that have essentially the same conditions and conclusions. Redundancy can be due to duplicate rules or the creation of equivalent rules (rule groups) by wording variations in the names given to variables, or the order in which they are processed. Subsumed rules occur when one rule's (or group of rules') meaning is already expressed in another rule (or group of rules) that reaches the same conclusion from similar but less restrictive conditions. Conflicting rules are rules (or groups of rules) that use the same conditions, but result in different conclusions, or rules whose combination violates principles of logic (e.g., transitivity). Circular rules are rules that lead one back to an initial (or intermediate) condition instead of a conclusion.

Logical Completeness. A knowledge base is complete if it has no holes or gaps in its logic. The following attributes indicate a logical incompleteness. Unreferenced attribute values are values on a condition that are not defined; consequently, their occurrence cannot result in a conclusion. Illegal attribute values are values on a condition that are outside the acceptable set or range of values for that condition. An unreachable conclusion is a conclusion that cannot be triggered by the rules combining conditions. Dead ends are rules that do not connect input conditions with output conclusions.

Content

Functional Completeness is the extent to which the knowledge base addresses all domain conditions. All desired inputs: the knowledge base can handle all input conditions that need to be addressed. Application/conclusion completely covered: the knowledge base can trigger all output conclusions that need to be addressed. Identified knowledge

limitations: the rules in the knowledge base can tell the user if input conditions currently being processed cannot be addressed. Analogously, if the expert system is such that a user can specify a conclusion in order to identify the input conditions that would generate it (e.g., as in a backward-chaining system), an expert system that was knowledgeable of its limitations would tell users if a conclusion currently being processed as input could not be addressed.

Predictive Accuracy. The following attributes address the accuracy and adequacy of the knowledge base. Problems here may also be related to problems of performance. Accuracy of facts: the quality of the unconditional statements in the knowledge base. Accuracy of rules: the quality of the conditional statements in the knowledge base representing expert judgment. Knowledge representation acceptability: whether or not the scheme for representing knowledge is acceptable to other domain experts and knowledge engineers. Adequacy of source: the quality of the persons or documentation used to create the knowledge base. Modifiability of knowledge base: the extent to which the knowledge base can be changed and the control over that change.

INFERENCE ENGINE: the extent to which the inference engine provides error-free propagation of rules, frames, probabilities, or other representation of knowledge or uncertainties used in the system.

"SERVICE" refers to aspects of the system (computer and others) in which the expert will operate.

Computer System. Design: the extent to which the expert system runs on the approved computer hardware and operating system and utilizes the preferred complement of equipment and features. In some cases, the design system will be stated in a requirements document; in other cases, the tester may need to survey available equipment at the intended installation. Portability: how easily the expert system can be transferred to other computer systems.

Computer Usage. Set-up time: the amount of time required for the computer operator to locate and load the program (if any) and the time to activate the program. Set-up time should be measured under the expected operating conditions. Run time: the amount of time required to run the program with a realistic set of input data. This attribute refers only to the time that the computer program takes to run; the time needed for the user is under PERFORMANCE factors. Space requirements: the amount of RAM, disk, or other space required by the program. Hardware reliability: the percentage of time the computer system could be expected to be operating effectively. Hardware capability: the computer system's total amount of RAM, disk, or other space. Effect of feature use/jumping: the extent to which moving from various parts of the program causes errors. Degradation: how well the program saves data and analyses and permits continuation after an unexpected program or system crash or power outage. Handling input errors: the extent to which the program prohibits a program crash and tells the user what to do after an input mistake.

System Integration. Formats: the extent to which the program uses input and output formats that are consistent with the intended use. This includes any mandated or standard formats that are specific to the intended user organization. Data requirements: the extent to which the program's data requirements are consistent in content, quantity, quality, and timeliness with those available to the intended user organization. The expert system should also be able to interact with specified and appropriate databases and communications systems. Documentation: the adequacy of material regarding the program's use and maintenance. Copies of computer code and its supporting documentation should be complete and understandable, and should allow maintenance by the user organization. (All applicable software documentation standards should be met.) Skill requirements: the extent to which the program can be operated by appropriately skilled individuals. The appropriate skill requirement includes grade level (for military enlisted, military officer, or civilian personnel), users' technical background, and training requirements. The appropriate level may be specified in requirements or may be determined by reference to the organizational setting of its intended use and to the personnel assigned to that setting.

PERFORMANCE refers to the operation of the expert system and the user. It includes both comparisons with ground truth and judgmental assessments.

Performance against Ground Truth. Speed: the amount of time it takes a user working with the expert system to solve representative problems. Accuracy: the degree of overlap in the distributions of belief values when the hypothesis is true versus false (see Chapter 5). Bias: the difference in the proportion of false negatives (hypothesis is true but system says false) to false positives (hypothesis is false, but system say it's true) (see Chapter 5).

Judgmental Performance. Response time: the judgments of users regarding the adequacy of the amount of time the expert system takes to react to inputs. Time to accomplish task: the judgments of users regarding the adequacy of the amount of time required to perform the task when using the expert system. Quality of answers: the judgments of users and experts regarding the system's capability. Quality of reasons: the judgments of users and experts regarding the adequacy of the system's justification for its answers.

USABILITY is the extent to which the expert system, or parts of the expert system, is used, is acceptable to individuals, and is acceptable to the organization.

Observable Usability includes aspects of usability that a tester can observe (or a system can record) during a test without asking the test subjects. Extent of use: how much users employ the expert system to perform the task (e.g., the proportion of time that the system was used to accomplish tasks assigned in a test). Manner of use: the way in which users employ the system and its features, including the procedures to access different modules, the way that intermediate and final outputs are incorporated into the user's results, and the use of interfaces. Features used: the extent to which different aspects of the expert system are employed by users.

Opinions about Usability. Confidence: how confident users feel in taking actions based on working with the expert system. Ease of use: how easy users judge the system is to use after they have completed training and become familiar with the system. Acceptability of person/machine interaction process: the extent to which users assess that they and the system are performing the tasks or activities for which they are best suited. Acceptability of results: the users' judgments regarding the adequacy of the system's capability. Acceptability of representation scheme: the users' judgments regarding the adequacy of the system's way of presenting knowledge. Input/output: the user's judgment about the adequacy of the extent, display, and manner of accessing the expert system's input and output.

Scope of Application: the users' judgments regarding the adequacy of the expert system in addressing domain problems.

Explanation. Adequacy of presentation and trace: the users' judgments regarding the acceptability of the system's presentation of its reasoning process. Transparency of expert system: the extent to which the system's reasoning process is clear and understandable to its users.

Organizational Impact. Impact on work style, workload, skills, and training: the judgments of users regarding the impact of the expert system on how they do their job, or the skills and training required to perform it effectively. Impact on organizational procedures and structure: the judgments of users regarding the impact of the expert system on the organization's operations.

Measurement Scales for Attributes

Appropriate scales for the attributes may differ from one expert system to another. Although it is impossible to set scales that will apply to every expert system in every operating condition and every intended use, we can suggest scales that the tester should consider. These are given below. Some suggested scales are simple "Yes or No" categorizations, others are natural units such as minutes, still others are percentages.

These may be helpful in establishing consistent frames for assessing the performance of a system that is being tested. We have avoided guidance on specific criteria of acceptability (e.g., "set-up time should be less than 10 minutes") because such criteria depend critically on specifics of the expert system and its intended use. We feel that generalizations of this nature would not be supportable. In general, these scales should be set before a test is begun. In addition, the relationship between performance on the scales and the utility of that performance should also be established, for example as discussed in the following section of this chapter.

KNOWLEDGE BASE

- *Logical Consistency*:
  - *Redundant Rules*: Percentages. The tester will examine the rule base and determine the percentage of individual rules and rule sets that are redundant. The tester may be able to perform a manual walk-through of small rule bases, but use of multiple software testers is better because the tedious nature of the task will no doubt result in errors. If an automated "static tester" were not available for a large rule base, some sampling procedure would be required.
  - *Subsumed Rules*: Percentages. Same rationale as that presented for "redundant rules."
  - *Conflicting Rules*: Number. Our definition was that conflicting rules used the same (or very similar) initial conditions, but resulted in either different conclusions, or violations in logic. In contrast to redundant or subsumed rules, which affect system efficiency, conflicting rules could well result in bringing the system to a halt unless there is an effective conflict resolution mechanism; at the least, it results in a logic error. Unless the initial conditions for conflicting rules are extremely rare, even 1 or 2 conflicting rules (or rule sets) that essentially crash the system may be unacceptable even though their percentage in the rule base may be extremely small.
  - *Circular Rules*: Number. Same rationale as for "Conflicting Rules."
- *Logical Completeness*:
  - *Unreferenced Attribute Values*: Number, because the effect is on system effectiveness, not efficiency. (This assumes that the unreferenced attribute values could occur in the operational environment. If they cannot, then they are more like

"Unnecessary If Conditions," affecting the efficiency with which the system examines the rule base.)

- *Illegal Attribute Values*: Number. Same rationale as for "Unreferenced Attribute Values."

- *Unreachable Conclusion*: Number. Same rationale as for "Unreferenced Attribute Values."

- *Dead Ends*: Number. [Note: Effectiveness vs. efficiency concern.]

- *Functional Completeness*:

  - *All Desired Inputs*: Number. This again addresses effectiveness. It should be remembered that this "Functional Completeness" assessment is made by reference to a requirements statement, or, if that does not exist, by domain experts. Consequently, each violation on this attribute may need to be examined because even one or two input omissions may have a significant impact on the utility of the system. The tester should consider placing a threshold of "no omissions" on this attribute.

  - *Application/Conclusion Completely Covered*: Number. Same rationale as for "All Desired Inputs."

  - *Identified Knowledge Limitations*: Yes or No. Most likely, the expert system either claims to have this capability or it does not, and the feature either works or it does not.

- *Predictive Accuracy*

  - *Accuracy of Facts*: Number. Each "inaccurate fact" needs examination in order to assess the utility score on this attribute. Accuracy should be determined by reference to an acknowledged source.

  - *Accuracy of Rules*: Number. This can usually be determined only by an expert or, preferably, by a group of experts. Each "inaccurate rule" needs to be examined to assess utility score on this attribute.

  - *Knowledge Representation Acceptability*: Yes or No. The implemented knowledge representation scheme is acceptable or not to other domain experts and knowledge engineers. The tester may want to get the opinions of several knowledge engineers and domain experts, if possible, for this assessment. "Other" knowledge engineers might conclude, on either effectiveness or efficiency grounds, that (a) an inappropriate representation scheme was used, or (b) that an appropriate scheme was not implemented well. Such assessments may be particularly important when the expert system is in the prototype stage.

- *Adequacy of the Source*: Yes or No. It is possible for a source to provide accurate information, but for it to be so limited as to be inadequate. This attribute will most likely require the opinions of a domain expert or panel of experts.

- *Modifiability of Knowledge Base*:

  -- *Control Over*: Yes or No. A software tester can assess whether accessibility to the knowledge base is controlled or not. A requirements statement, sponsoring agency, users, and perhaps security analysts and domain experts may be needed to assess whether the level of control is acceptable or not.

  -- *Expandability* (by human/machine): Yes or No. Again, a tester can assess whether the knowledge base can be increased (i.e., expanded), decreased or, in general, modified by humans and, perhaps most interestingly, by machines. A requirements statement (or the system's sponsoring agency) may provide an assessment of whether such expandability is desirable. Domain experts working with AI specialists would probably be required to assess whether the changes were acceptable. [Note: Acceptability, in terms of performance, could be determined by statistical analysis of test cases where subjects changed the knowledge base.]

"SERVICE"

- *Computer System*:

  - *Design*: Yes or No. Consistent with the definition, the expert system either runs on the approved computer hardware and operating system (and utilizes the preferred equipment and features) or it doesn't. If it does, then it passes. If it doesn't, then it fails; the utility score (e.g., between "0" and "50") would depend on the type of incompatibility problems found by the software tester. [Note: If the system scores "0" on "Design," which means that it does not run on the approved hardware and operating system, then its values for "Set-Up Time," "Run Time," "Space Requirements," etc. are all tied to the hardware the system does run on. For an early prototype, this may be quite acceptable, for "Design" may have a low weight. However, in the later stages of development, there may be a noncompensatory threshold rule where a "0" on "Design" results in an unacceptable score overall.]

  - *Portability*: Yes or No for comparable machines. For example, if the expert system was developed for an IBM AT, then it would "pass" on portability if it could run on AT-compatibles of similar power. If it could also run on an IBM PC (or compatibles), then it would get a utility score greater than "50," depending on whether it ran with all its features. If it

couldn't run well on an AT-compatible, it would receive a score less than "50." If it couldn't run at all on an AT-compatible (or a PC), it would get a score of "0." The same logic holds for mainframes, and for going between mainframes and personal computers. [Note: It is possible that the system is portable with one type of hardware, but not another. The tester should refer to any statement of requirements to determine the range desired. The hardware "types" would receive weights to obtain a total score.]

- *Computer Usage*:

  - *Set-Up Time*: Minutes. The software tester may want to calculate the average and standard deviation for this attribute. However, that requires that the software tester perform the set-up a number of times (e.g., 10). The amount of time required for such repetition, particularly for measuring other attributes (e.g., "Run Time" or "Ground Truth Performance") is probably unacceptable unless the attribute is very important.

  - *Run Time*: Minutes. The tester should record this for all test cases (to the extent possible) and may use statistical summaries (e.g., mean and standard deviation) in the assessment.

  - *Space Requirements*: The amount of RAM and disk space required to run the system. Standards of acceptable size may be stated in a requirements document. Otherwise, acceptable sizes might be determined by the tester based on the total available.

  - *Reliability (Hardware)*: Percentage of time in a 24-hour day (or during specified periods) that the computer (i.e., hardware) is operating effectively. [Note: The tester might want to expand the definition to include software if the expert system requires distributed databases that require periodic updating and possible "down time," independent of the hardware.]

  - *Capability (Hardware)*: The computer system's total amount of RAM and disk space. The importance of this will be related to how close the expert system comes to using all available space.

  - *Feature Use/Jumping*: Number (and type). Each case where moving from one part of the program to another caused an error would have to be examined because of its potential effect on system effectiveness.

  - *Degradation (Graceful?)*: Number (and type). The concern is on the effect of ungraceful degradation on effectiveness. In some operational environments, even one ungraceful degradation would be unacceptable. This attribute might also be measured on a "Yes or No" scale on the assumption that the system should degradate gracefully, regardless of the cause precipitating the system crash or power outage.

- *Handling Input/Output Errors*: Number (and type). Same rationale as for "Degradation (Graceful?)." This could also be "Yes or No" on the assumption that the system could (or couldn't) tell the user what to do after an input mistake, but it's possible that this capability could exist in some modules and not others.

- *System Integration*

  - *Formats*: Number (and type). Identify all inconsistencies with input and output formats specified in the requirements document or other appropriate source.

  - *Data Requirements*: Number (and type). Identify all inconsistencies in the content, quantity, quality, and timeliness of the system's data requirements and those specified in the requirements document or other appropriate sources.

  - *Documentation*: Acceptable or Unacceptable. All applicable DoD software documentation standards were met. Standards that were failed should be identified by the software tester. If DoD standards aren't appropriate, the software tester should assess whether the expert system's documentation is, in general, complete and understood or not. Problem areas need to be identified. This assessment should be separately performed for (a) the user's manual, (b) the operator's manual, and (c) the computer code and its supporting documentation.

  - *Skill Requirements*: Yes or No. This may be difficult to assess. The concern is whether, prior to giving the system to users, software testers could make an initial assessment of whether targeted users have the required background skill to effectively operate the system. After examining the (1) requirements document and (2) documentation describing the users' organizational setting, this may be an easy or difficult assessment. The binary "Yes/No" measurement scale is a conservative scale. That is, passing the "Skill Requirements" attribute should be easy to assess or the system fails. For example, for one Army expert system, this proved to be a critical issue. The terminology used in the system was the terminology of the experts and proved beyond the entry level of the user actually causing the users to interact with the system in an incorrect manner. This was partially because the skill level of the users was based on completion of a certain course which no longer contained many aspects that were in the course when the experts took the course.

PERFORMANCE

- *Ground Truth*:

  - *Speed*: Minutes. Consistent with the previous discussions, software testers should calculate the mean and variance for the amount of time it takes the (test) users to solve (representative) problem scenarios working with the expert system.

  - *Accuracy (d*)*: Probability that two points, one taken from the Positive distribution (i.e., the hypothesis is true) and one taken from the Negative distribution (i.e., the hypothesis is false) will be in reverse order. That is, the probability that the belief value of a point x_p from the P distribution is lower than the value of a point x_n from the N distribution:

$$d^* = p(x_p < x_n | x_p \epsilon P,\ x_n \epsilon N).$$

    (See Chapter 5 for details.)

  - Bias (B*): Is calculated by the following formula:

$$B^* = \frac{\text{\# false alarms}}{\text{\# in } S_N} - \frac{\text{\# false positives}}{\text{\# in } S_P}.$$

    (See Chapter 5 for details.)

USABILITY

- *Observable*:

  - *Extent of Use*: Proportion of time the system was used for task accomplishment. Again, propose calculation of the mean and variance for this distribution.

  - *Manner of Use*: Type and Percentages. The software tester identifies the different ways in which users employed the expert system and its features. Then the tester calculates the percentage of users who used the system in each of the identified ways.

  - *Features Used*: Percentages. Tester calculates the percentage of users who used each of the system's basic features when solving the problem scenario.

Judgmental Performance and the Rest of Usability

Two forms of questionnaires are provided for these attributes in the Appendix. These questionnaires should be used with a sample of test subjects and the means and variances calculated for assessing performance on the attribute.

Using the Hierarchy for Testing

As discussed earlier in this chapter, several steps are necessary to use the hierarchy of attributes: identify what is being tested, establish importance weights, define measurement scales for the criteria, convert the scales to a common unit of utility, test the expert system against all important criteria, and combine the results into an overall assessment. In this section, we present some additional suggestions for performing these steps. Volume 5 contains a detailed example of the method implemented with TESTER_C, the computer program that incorporates the MAU hierarchy.

In many cases, it will be useful to construct hypothetical "benchmark systems," in addition to the expert system being tested, to use as points of reference. The tester may want to consider the following "systems":

- the test system, which is the expert system being subjected to testing;
- a goal system, which is a hypothetical system that fully attains every goal on every attribute;
- a failing system, which is a hypothetical system that fails on every attribute;
- a marginal system, which is a hypothetical system that, on balance, would just manage to pass the test, considering its performance over all attributes.

Introduction of these hypothetical systems enables a tester to apply the test criteria on a consistent, comparative basis, and to highlight areas of deficient and superlative performance of the expert system being tested. Of the hypothetical systems, the marginal one is usually the most difficult but most important to describe. Any given system under test is

likely to have some areas where it falls short of goals and others where it exceeds goals. In addition, some of the goals may be set as ideals that could not be expected to be met. The marginal system provides a way for the tester to interpret performance in a way that recognizes these possibilities, and to specify *in advance* a minimal level of acceptable overall performance. This specification in advance removes some of the subjectiveness of the process by setting an overall level of acceptability before test results are known. Note that the marginal system will not generally be unique. Many possible combinations of performance against attributes may be minimally acceptable. However, when the MAU model is fully specified, all of these marginal systems should receive about the same overall evaluation (i.e., weighted utility). Specification of one of these systems thus aids in the overall evaluation of the actual system being tested.

To convert measurement scales on attributes—such as those suggested in the section above—to a common utility scale, we suggest the following 0 to 100 point scale. A convenient, consistent scale could assign a 0 to performance that is a failure against the attribute, and a 50 to performance that meets the performance goal fully. This choice is arbitrary in the sense that these levels of performance could be assigned any numbers, for example, 0 and 100, 0 and 1000, or 27 and 78. However, the points are not arbitrary in their meaning; 0 is assigned consistently to the failure level, and 50 is assigned consistently to the level of full satisfaction. This assignment provides a basis for consistent interpretation of the analysis and provides the kind of consistency that reduces bias from the assessments. The scale also allows value to be attached to performance that exceeds the goal, by scores greater than 50. A score of 100 is used, for example, in the questionnaires on subjective attributes to represent a performance that greatly exceeds the goal.

The scales represent ratio judgments of value in the following manner. A score of 25 is half-way (in value) between failure and full goal attainment. This provides for convenient and consistent interpretation of scores. However, it is left to the tester to define the levels of performance that represent the goal and failure, and these will change from situation to situation. For example, a 20-minute set-up time may meet the

goal fully in some cases but may fail in other cases. Another assessment that the tester must make is whether the performance on any single attribute is so important that the expert system would be regarded as a complete failure if it failed on that attribute, *regardless of its performance on all other attributes*. In this case, a threshold of performance should be applied to this attribute.

These utility values represent value on individual attributes only, and a value on one scale is not generally comparable to a score on another attribute, except by reference to the goals. The weighting procedure described earlier in this chapter provides a means for comparing across attributes. The weighting procedure assigns a relative importance to criteria and attributes in the hierarchy. Such assessments are usually best made by reference to a requirements document or by asking a responsible individual or organization, such as the sponsoring agency, directly. In some cases, the tester may have to infer this information from available information. For example, this process may start by asking the sponsoring agency to assign relative importance to

- Structure and Control of the Knowledge Base;
- Performance of the Inference Engine;
- "Service" Aspects;
- Performance of the Expert System; and
- Usability.

(Remember that the question refers to the importance of the range of performance between failure and the goal level on all attributes beneath each category.) The tester might then use his information and understanding of the system to assess the relative importance of sub-categories, for example the relative importance of structure and content of the knowledge base. For the more important categories, it is best to try to extract as many of the sub-category tradeoffs from the sponsoring agency or the requirements document as possible. This process is continued until relative weights are assessed for all attributes.

Against this structure, the performance of the expert system is assessed against each attribute using the appropriate subjective, technical, or empirical methods discussed in this book. (Similar assessments may be made judgmentally for any hypothetical systems used.) Next, an overall assessment is determined by the weighted-averaging technique described in this chapter; that is, assessments of performance on the attributes are converted to utilities which are multiplied by the appropriate weights and summed (see Equation [3-1]). This score for the system being tested is then compared with those for the hypothetical passing and marginal system and checked against any threshold attributes for an overall assessment. (Remember also that a score of 50 overall is interpreted as meeting the goal.)

CHAPTER SUMMARY

This chapter provided a detailed overview of five subjective evaluation methods: multiattribute utility assessment (MAUA), cost-benefit analysis, the dollar-equivalent technique, decision tree analysis, and a MAUA-based cost-benefit analysis. All five methods are oriented to testing and evaluating expert systems. With MAUA, utility functions and a weighted, additive decision rule are typically used to convert the system's scores on multiple attributes (or evaluation criteria) into a single, global measure of effectiveness called an overall utility value. With cost-benefit analysis and the dollar-equivalent technique, dollars are used instead of utilities to represent the overall worth of the item. Specifically, cost-benefit analysis uses standard accounting practices, such as rate of return and time value of money, to create monetary equivalents; the dollar-equivalent technique develops the equivalents judgmentally when the standard practices are stretched beyond their limit. Decision tree analysis provides an explicit, formal method for combining uncertainties with utilities (or dollar equivalents) when evaluating a system. The best item is the one with the highest "expected utility" or "expected value," depending on whether one is using utilities or dollar equivalents. Finally, the MAUA-based cost-benefit analysis uses cost-benefit ratios, where benefit is defined in terms of a utility scale to evaluate items. Conceptually, it could also be done for dollar equivalents.

All five subjective evaluation methods are applicable throughout the expert system development process. The appropriateness of a method depends on the information and decision needs of members of the sponsoring or development team, available time and resources, and the comfort that participants have with the method. Moreover, one should note that "information need" and "comfort" are conceptually independent. For example, decision makers typically feel more comfortable with "objective, quantitative" measures of effectiveness (MOEs). However, they also need to consider more qualitative, subjective MOEs—such as user preferences—when assessing expert systems. All five subjective evaluation methods are capable of handling objective and subjective MOEs.

Next, we presented methods for constructing questionnaires to elicit opinions. This included basic methods for developing question and response scales, arranging questions in a questionnaire, assessing the reliability and validity of a questionnaire, and alternative types of questionnaires. A generic questionnaire for assessing performance and usability characteristics in a test of an expert system was developed following these guidelines and is presented in the Appendix to the book.

Finally, the last section of the chapter proposed a particular MAU framework for testing and evaluating expert systems. It described and defined a hierarchy of attributes, suggested measurement units for the attributes, and provided guidance on using the hierarchy in testing.

CHAPTER 4:

MORE ABOUT TECHNICAL TEST AND EVALUATION METHODS

Technical test and evaluation methods assess how well the system was built. In particular, the focus is on (1) the logical consistency and adequacy of the knowledge base, (2) the functional completeness and predictive accuracy of the knowledge base, (3) the adequacy of the inference engine, and (4) the general speed and compatibility of the system's software and hardware for the organizational setting where it will reside.

Chapter 4 overviews three classes of technical test and evaluation methods: (a) static and dynamic testing methods for assessing the logical consistency and completeness of the knowledge base and the adequacy of the inference engine; (b) methods for using domain experts to assess the functional completeness and predictive accuracy of the knowledge base; and (c) conventional software test and verification methods for assessing the service requirements of the entire system. The three sections in this chapter are organized around the first three criteria in the MAU hierarchy presented in Table 2-1. In particular, the first section deals with testing and evaluating the expert system's knowledge base. The second section considers the test and evaluation of the inference engine, and the third section, the expert system's service requirements. Each is now considered, in turn.

TESTING AND EVALUATING THE KNOWLEDGE BASE

Consistent with the MAU hierarchy in Table 2-1, the knowledge base sub-branch is divided into two groups of criteria—those focusing on the adequacy of the knowledge base's structure and those focusing on the adequacy of its content. For structure, we consider attributes that address the logical consistency and logical completeness of the knowledge base. For content, we consider attributes that address the functional completeness and predictive accuracy of the knowledge base.

In general, there are two classes of technical test and evaluation methods for assessing knowledge base quality. First, static testing

methods, and, to a lesser extent, dynamic testing methods, can be used to assess the logical consistency and completeness of the knowledge base. Second, domain experts working with test cases and employing empirical test and evaluation concepts and methods can be used to assess the functional completeness and predictive accuracy of the knowledge base. These two classes of methods are discussed below within the context of assessing the adequacy of an expert system's structure and content.

Methods for Evaluating Logical Consistency and Completeness

Some authors (e.g., Rushby, 1988) have suggested that techniques for static testing of conventional software are readily extended to expert system knowledge bases. In both cases, the focus is on detecting anomalies without execution. In expert systems these anomalies relate to the logical consistency and completeness of a knowledge base. In the last few years, a number of researchers have developed static testing methods that can be applied to expert system rule bases (Nguyen et al., 1987; Stachowitz and Combs, 1987; Franklin et al., 1988; Gilbert, 1989). In this section we will examine three of these techniques, plus Kang and Bahill's (1990) dynamic testing approach embodied in their software called "Validator."

Static Testing for Categorical Expert Systems. A categorical system is one that reasons qualitatively—it does not consider gradations of belief. Typical of such systems are rule-based expert systems. As noted in previous chapters, these systems process through chains of if-then rules to generate conclusions. If one treats if-then rules as logical expressions, then it is possible to check these rule sets for a variety of logical errors. Examples of such errors (from Adelman and Ulvila, in press) include those listed below. More extensive taxonomies can be found in Kirk and Murray (1988), Nazareth (1989), and Rushby (1988).

- *Redundant Rules.* Individual rules or groups of rules that essentially have the same conditions and conclusions.
- *Subsumed Rules.* When one rule's (or rule group's) meaning is already expressed in another's that reaches the same conclusion from similar, but less restrictive, conditions.

- *Circular Rules*. Rules that lead one back to an initial (or intermediate) condition(s) instead of a conclusion.

- *Unreferenced Attribute Values*. Values on a condition that are not defined; consequently, their occurrence cannot result in a conclusion.

- *Illegal Attribute Values*. Values on a condition that are outside the acceptable set of values for that condition.

- *Unreachable Conclusion (and Dead Ends)*. Rules that do not connect input conditions with output conclusions.

Static testing for the above anomalies could be performed manually for small, well-structured knowledge bases. For even moderately sized knowledge bases, however, this approach is precluded by the amount of effort required and the probability of disagreements among testers. Consequently, researchers (e.g., Culbert and Savely, 1988; Franklin et al., 1988; Nguyen et al., 1987; Stachowitz et al., 1988) have begun developing automated static testers. Although a number of different technical approaches are being investigated, Gilbert (1989, p. 2) has noted that many of the automated static testers "... either implicitly or explicitly consider an expert system's rule base to be a graph or network. In the graph of a rule base, there are nodes that represent rules and nodes that represent the hypotheses that appear in the rules' premises and conclusions. There is an arc from a hypothesis to each rule whose premise it appears in. There is an arc to a hypothesis from each rule that asserts the hypothesis in its conclusion." Thus, a graph can represent the knowledge base's logical structure (and flow) and, thereby, help detect the types of logical consistency and completeness errors defined above.

Although a valuable pictorial display, a graphical representation of even a moderate-size rule base can be difficult to use for error detection. Consequently, researchers have begun using matrices and Boolean algebra to automate the error detection process. To illustrate this, we use an example developed by Bellman and Walter (1988) to represent a common source error, which is when the same piece of information goes into different lines of reasoning. Specifically, Figure 4-1 shows the flow graph for a fragment of a fictitious knowledge base for diabetes diagnosis.

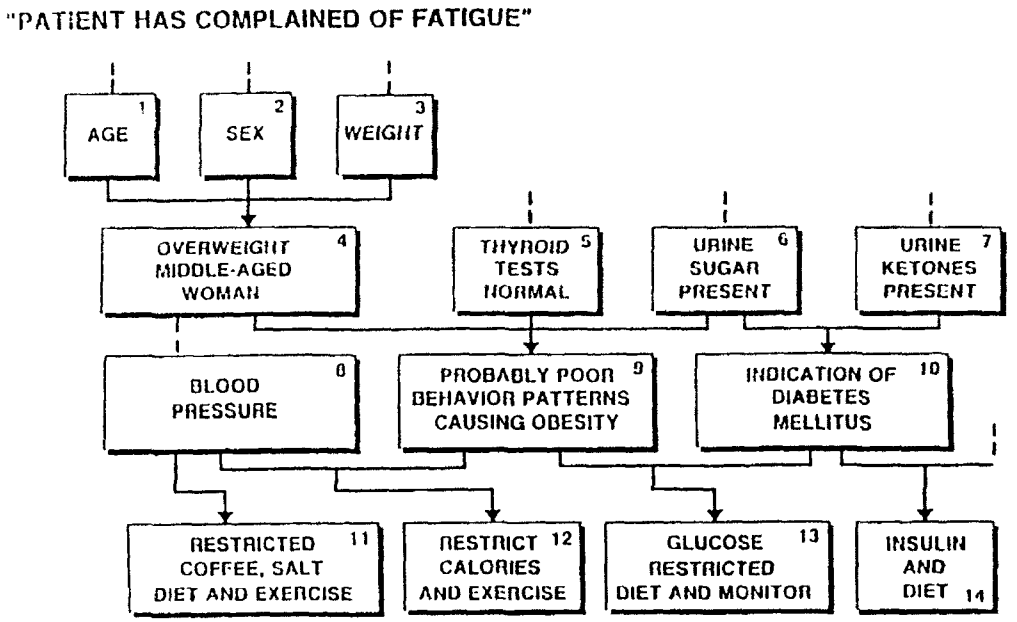


Figure 4-1: The Flow Graph for a Fragment of a Fictitious Rule Base for Diabetes Diagnosis

Figure 4-2 presents an incidence matrix for representing this graphic flow. The rows in an incidence matrix represent inputs; the columns represent outputs; and the "1s" represent the connection. For example, the information obtained for age, sex, and weight (i.e., boxes 1, 2, and 3) only goes into box 4, which, in the example, is that the patient is an overweight, middle-aged woman. Information from boxes 4, 5, and 6 goes into box 9, and so forth.

into: / out of\	4	9	10	11	12	13	14
1	1						
2	1						
3	1						
4	*	1					
5		1					
6		1	1				
7			1				
8				1	1		
9		*			1	1	
10			*			1	1

Figure 4-2: An Incident Matrix Representing the Flow Graph for the *Fictitious Diabetes Diagnosis* Rule Base

The next step is to translate the incidence matrix into Boolean polynomials. This translation process in this example depends on the truth table shown in Figure 4-3.

Assertion:	4	5	6	9
Rule				
R 41	F	F	F	F
R 42	F	F	T	F
R 43	F	T	T	F
R 44	F	T	F	F
R 45	T	F	F	F
R 46	T	F	T	F
R 47	T	T	T	F
R 48	T	T	F	T

Assertion	6	7	10
R 61	F	F	F
R 62	F	T	F
R 63	T	F	F
R 64	T	T	T

Assertion	9	10	13
R 91	F	F	F
R 92	F	T	F
R 93	T	F	F
R 94	T	T	T

Figure 4-3: A Truth Table Representing a Fragment of the Fictitious Rule Base for Diabetes Diagnosis

We quote Bellman and Walter (1988, p. 7) to illustrate this process.

> We will use "4" to mean "is an overweight, middle-aged woman," while 4' means "is anything else." Similarly, "5" means "thyroid tests normal" while 5' means "thyroid tests abnormal;" and so on. In that notation,
>
> $$9 = 4 * 5 * 6'$$
>
> where we use * for Boolean product (some other notations use ^). Further,
>
> $$10 = 6 * 7$$
>
> $$13 = 9 * 10.$$

But we can substitute into the expression for 13:

$$13 = 9 * 10 = (4 * 5 * 6') * (6 * 7) = 4 * 5 (6' * 6) * 7 = 0,$$

where we show ( ) around the factors which create the 0 product. The conclusion that 13 = 0 means that 13 can never be set to "T." Another way of stating this is that the rules assigning 13 the value T [True] can never be utilized.

This mathematically induced result can be seen by examining the truth table shown in Figure 4-3. Rule 94 (R94) asserts that 9 and 10 must be true for 13 to be true. Rule 48 asserts that 9 is true if 4 is true, 5 is true, and 6 is false. However, Rule 64 says that both 6 and 7 must be true for 10 to be true. Since 6 cannot be true and false at the same time, a logical flaw in the knowledge base has been discovered.

Approaches similar to the incidence matrix approach can be found in Sheppard, 1989.

For instance, Nazareth (1989) has shown that if a set of rules contains a contradiction, then a logical statement of the form

$$X \vee X \vee Z1 \vee \ldots \vee Zn$$

can always be derived using a theorem-proving technique called resolution refutation. [Note, $\vee$, &, $\rightarrow$ means "or," "and," and "if..then.." respectively.]

To illustrate this approach, begin with the following rule set. Although not apparent at first look, this rule set contains a contradiction.

R1 $A \& B \rightarrow C \& \neg D$

R2 $C \& E \rightarrow \neg F$

R3 $\neg D \rightarrow F \& \neg B$

R4 $E \rightarrow A$.

To perform resolution refutation, one first translates logical expressions into clause form, which is simply a list of disjunctive expressions. A clause form database that is logically equivalent to the above four rules is the following.

C1 ~A ∨ ~B ∨ C

C2 ~A ∨ ~B ∨ ~D

C3 ~C ∨ ~E ∨ ~F

C4 D ∨ F

C5 D ∨ ~B

C6 ~E ∨ A.

The standard technique for making deductions from a clause database is resolution. Resolution applied to two clauses of the form X ∨ Y1 ∨ ... ∨ Yn and ~X ∨ Z1 ∨ ... ∨ Zn results in the resolvent clause Y1 ∨ ... ∨ Yn ∨ Z1 ∨ ... ∨ Zn. Resolvent clauses are logical deductions that can be added to the database, from which more clauses can be deduced. Using resolution, C2 and C5 resolve to generate the clause

~A ∨ ~B ∨ ~B.

Since ~B appears twice, the rule set must contain a conflict. (Specifically, given A & B, R1 will conclude ~D from which R3 will conclude ~B.) In a similar manner, any possible circular or redundant rules, unreachable conclusions, or unnecessary if conditions can be discovered using resolution refutation (see Nazareth, 1988, for a complete discussion).

As these examples illustrate, there are a variety of techniques for identifying anomalies in a rule base, and simple pairwise comparison of rules is not always sufficient. Many of these techniques are extendable to other categorical knowledge representation schemes. For instance, the logic-based approach can be applied to any knowledge representation scheme where the knowledge base can be converted into a set of expressions in symbolic logic. Examples of such schemes include frames (Hayes, 1981), scripts, semantic nets (Nilsson, 1980, ch. 9), etc.

Several research tools have been developed to check knowledge bases' (especially rule bases') consistency. CHECK (Nguyen et al., 1987) was developed for testing knowledge bases built for the Lockheed expert system shell, LES. EVA (Stachowitz and Combs, 1987), the Expert system Validation Associate, was developed for ART and could be extended for other shells. The Expert System Planning Environment, ESPE (Franklin et al., 1988), was developed at Rensselaer Polytechnic Institute for IBM's Expert System Development Environment. The Expert System Examiner, ESE (Gilbert, 1989), was developed by Booz, Allen & Hamilton for ESL (a Lisp-based expert system language), a pseudo-English, and Nexpert Object. These systems, however, are research tools that were developed for limited distribution and use. There are currently no plans to offer these tools as commercial products or to otherwise make these tools available to testers. Unfortunately, a tester who wants to automate the static testing of a knowledge base will probably have to develop his own tool for doing so.

Static Testing for Systems with Reason Maintenance. One approach to managing uncertainty is to make assumptions, and then to reason from those assumptions. The capability of reasoning from and retracting assumptions is variously referred to as truth maintenance, reason maintenance or assumption-based truth maintenance. We will simply refer to it as assumption-based reasoning.

Assumption-based reasoning is becoming commonplace in expert system development. Most of the more sophisticated shells now have this capability.

Assumption-based reasoning complicates the problem of static testing. This is because these systems are designed to reason consistently in the context of contradictory conclusions. Conflicting rules in a normal rule base may be perfectly consistent in an assumption-based system.

To illustrate, consider the following rule set. Here a, c, d, and f are propositions, B and E, assumptions.

R1 a → B

R2 B → c

R3 d → E

R4 E → f

R5 c → ~f.

Assume that a and d are entered as facts. Applying these rules, this rule base quickly concludes c & f—which contradicts R5. However, since this rule set contains two assumptions, B and E, a contradictory conclusion does not necessarily imply an inconsistent set of rules. One can always retract an assumption. In particular, an assumption-based system would simply deduce that there are two possible states {a,B,c,d,~E,~f} and {a,~B,~c,d,E,f}.

Static testing of knowledge bases with assumptions, therefore, is a somewhat different problem than simpler knowledge-based systems. Although some of the same criteria still apply (e.g., unreferenced attribute values), others do not. In particular, tests for logical consistency are very different. Currently, we do not know of any general techniques for static testing of systems with assumption-based reasoning.

Static Testing for Uncertain Inference Systems. Another approach to handling uncertainty is to use some type of uncertainty calculus for quantifying degrees of belief. When an uncertainty calculus is introduced, however, one quickly discovers that rule sets that would be considered logically inconsistent in a categorical system are perfectly consistent in a quantitative system. To illustrate this difference, assume that we have an expert system where the knowledge base corresponds to a set of probability statements. Consider the following rules:

R1 A & B → C (.7)

R2 C → D (.8)

R3 D → E & F (.65)

R4 A → ~F (.75)

R5 A → F (.25)

R6 F → A (.4).

Each of these rules is interpreted as a conditional probability statement. For example, the rule C→D (.8) simply asserts P(D|C)=.8. According to the taxonomy of anomalies presented earlier, this rule set is full of logical errors. Yet, from the perspective of the probability calculus, it is completely consistent. Rules R4 and R5, for instance, do not represent a conflict, but are two equivalent probability statements (i.e., probability P(~F|A) = 1-P(F(A)). Similarly, R1-R4 demonstrate that transitivity of inference does not hold when quantitative uncertainty is introduced. Given B, for instance, we know that A implies that C is probable, C implies that D is probable, and D implies that F is probable. If probability were a logical property, we would also conclude by transitivity that A implies that F is probable. In this rule set, however, we have the opposite assertion, namely that A implies that F is improbable. [A verbal example might help. Most college students are unemployed (P(unemployed|student) > .5). Most unemployed people have less than a high school education (P(uneducated|unemployed) > .5). If we insist on transitivity, we would also conclude that most college students have less than a high school education.]

The reason for the difference between categorical and probabilistic rules is that the probability calculus has its own consistency criteria that have little to do with "logical" consistency. In the case of the probability calculus, we know that if a rule set satisfies the following properties,

$P(True) = 1$

$P(\sim A) = 1 - P(A)$

$P(A \vee B) = P(A) + P(B)$ if A and B are disjoint

$P(B|A) = P(B \& A)/P(A)$,

then it is consistent with the probability calculus, and no conflict can exist.

The same is true of any principled approach to uncertainty management (e.g., fuzzy calculus or belief functions). Each uncertainty calculus defines its own criteria for consistency—each somewhat unique.

A Dynamic Testing Approach: Validator. Kang and Bahill (1990) have developed a tool called Validator that uses test cases to assess the logical consistency and completeness of the knowledge base. The test cases can be real or imaginary. They can be developed by domain experts, knowledge engineers, users—or, preferably, all three—to increase the probability that most segments of the knowledge base will be exercised. All that is required is that the test cases embody, at least on the surface, valid preconditions. The goal is to record rule firings, not to assess predictive accuracy. The focus is on identifying which rules never fire and why.

Kang and Bahill (1990, p. 48) discuss the following two classes of problems that result in rules not firing: "failure due to false premises" and "failure due to cutoff." Considering the former, if the premise to a rule is not satisfied by any of the test cases, then the rule will never fire. This failure can be caused by any of the following three attributes in our hierarchy: unreferenced attribute values, illegal attribute values, and unreachable conclusions (and dead ends). If a rule has an unnecessary if condition, it will still fire if the remaining if conditions are necessary and sufficient.

"Failure due to cutoff" implies that the system always stops before a certain rule is reached; therefore, the rule never fires. This failure can be caused by the four logical consistency attributes: redundant rules, subsumed rules, conflicting rules, and circular rules. It can also be caused if, as in most backward-chaining systems, the system stops after finding a value with complete certainty. To quote Kang and Bahill (p.48), "Consider this set of rules [where cf is a confidence factor scaled between 0 and 100%]:

rule-1: if a = yes then c = 1 cf 100%

rule-2: if a = no then c = ? cf 100%

rule-3: if b = yes then c = 3 cf X.

Rule-3 will never succeed. After the inference engine has found a value with 100% certainty, it won't seek further values. (This example presumes the user is not allowed to answer 'unknown' when a value for a is queried.)" (This example also assumes that the rules are accessed in the order shown.)

As this example illustrates, none of the four logical consistency errors is the problem. Instead, the problem is either that: (1) both preconditions for a incorrectly reach a conclusion with 100% certainty and, therefore, rule-3 is never reached; or (2) b represents an unnecessary if condition and, more generally, rule-3 an unnecessary rule. Deciding which of these two possibilities is the cause of the problem is up to the domain expert and knowledge engineer. Validator's task is simply to identify that rule-3 never fired for any test case.

Validator also identifies rules that fire all the time. As Kang and Bahill (p. 48) point out, such rules are probably mistakes or are, perhaps, better represented as facts. "Of course, some control rules always succeed, and some rules will be designed for rare situations not exercised by the test cases at hand. Again, this technique is only advisory; the expert must make the final decision about the rule's correctness."

As this last quote illustrates, Validator leaves a lot of the evaluation of the knowledge base in the hands of the domain expert or derivatively to the tester or developer. Moreover, it does not test the knowledge base (for categorical systems) for the various types of logical consistency and completeness errors that the static testing methods described above do. Instead, it records which rules fired for which test cases, and provides cumulative statistics, most importantly, identifying which rules never fired. Finally, it is dependent on the test cases used in the assessment. If the sample of test cases does not exercise the full range of cases for which the knowledge base was developed, then many of the rules in the knowledge base will not fire. Therefore, one might argue that Validator does not provide as rigorous a test of the knowledge base's logical con-

sistency and completeness as do static testing methods. However, as we mentioned before, automated static testers are not currently available; Validator is.

Summary. Many expert systems, like conventional programs, are categorical and do not involve assumption-based reasoning. For such systems, many of the static testing procedures that are used to evaluate programs can be adapted to evaluating a knowledge base. On the other hand, expert systems that manage uncertainty, either by making assumptions or using an uncertainty calculus, are not amenable to the same type of testing. Different uncertainty calculi require different criteria for consistency. Each system must be evaluated according to its own criteria.

As a practical matter, one may argue that there is little to be gained from developing and applying general software tools for static testing of knowledge bases—there are too many different criteria. On the other hand, with the growing interest in testing, one would like to find future development shells that have embedded appropriate static testing procedures. If available, such tools could be used for automated static evaluation. If unavailable, then about all that can currently be done is some direct manual checking or custom development of specialized tools.

Performing an unaided static analysis of even a small knowledge base is a tedious task. In fact, we would recommend that two or more testers be involved in an unaided static analysis of even medium-sized knowledge bases because the tedious nature of the task without some automated assistance will no doubt result in errors and omissions. In addition, the use of sampling procedures would also probably be required to reduce the amount of work and, in turn, cost. Seen from this perspective, Validator, even with its limitations, appears to be a worthwhile tool for helping testers.

Methods for Evaluating Functional Completeness and Predictive Accuracy

By functional completeness we mean to address the range of domain-oriented questions, such as whether the knowledge base contains all desired input conditions and output conclusions, or even "knows" its knowledge

limitations. Some of these questions can be answered by domain references. However, the level of domain expertise desired for expert systems is typically not codified in such references. Indeed, Davis (1989) has argued that one of the major contributions of expert system technology is that it has forced the organization and codification of various disciplines.

Consequently, domain experts are usually required to assess the functional completeness of the knowledge base. This is typically done by having experts perform two activities. First, one has experts examine the knowledge base (premises, rules, and conclusions) and question the developers on the various conditions the system can handle or not. Second, one has experts use test cases—both actual and hypothetical—to exercise the knowledge base. One should remember, however, that the system's level of functional completeness depends on its stage of development and, most importantly, the domain requirements resulting from the requirements analysis. The DART expert system prototype, for example, considered only thirteen of forty-two possible activity nodes. This was quite acceptable in the sponsoring team's opinion because they conceptually placed a low relative weight on "functional completeness" at this stage of DART's development.

The predictive accuracy of the knowledge base pertains to the correctness by which the facts and rules (or whatever representation scheme) relates the conditions in the test cases to the system's conclusions. Such an assessment is essential for expert systems; otherwise, "garbage in" is literally "garbage out." In particular, we have identified the following five attributes in Table 2-1 that address the predictive accuracy of the expert system's knowledge base:

1. *Accuracy of Facts*: the quality of the unconditional statements in the knowledge base;

2. *Accuracy of Rules* (or whatever representation scheme): the quality of the conditional statements in the knowledge base representing expert judgment;

3. *Knowledge Representation Acceptability*: whether or not the scheme for representing knowledge is acceptable to other domain experts and knowledge engineers;

4. *Adequacy of Source*: quality of the person(s) and/or documentation used to create the knowledge base;

5. *Modifiability of the Knowledge Base*: extent to which there is control over changes to the knowledge base, and whether these are implemented by (selected) humans and/or the machine itself (through learning).

Conceptually, knowledge engineering is a measurement problem. For most of the problem domains for which we develop expert systems, we do not have objective, quantitative knowledge (e.g., in the form of ground truth) that we can, so to speak, take off the shelf and put into a system to solve a problem. Instead, we have to rely on experts to tell us, on the basis of their learned knowledge and experience, what information and relationships between this information are important in solving a problem (or performing a task) in the domain for which we are building the expert system.

It is not uncommon for experts to not only disagree in their conclusions, but in how they reached them. Moreover, there is considerable research (e.g., see Ebert and Kruse, 1978; Hoffman et al., 1968; Libby and Lewis, 1977) demonstrating that, under controlled settings, the predictive accuracy of different experts varies considerably. Indeed, the commonplace phrase, "Get a second opinion," heard in many professional problem domains for which expert systems are under development, such as medicine, is not only indicative of this fact, but an illustration of the larger measurement problem. For if one wants to increase the probability that one has uncovered some truth and not just the random or systematic error attributable to the measurement instrument (e.g., domain expert), one should apply the measurement principle of sampling over sources of variability (e.g., see Adelman, 1989; Hammond, 1948).

The measurement problem is made more complicated in knowledge engineering efforts because there are four additional sources of variability of the knowledge base: knowledge engineers, knowledge representation schemes, knowledge elicitation methods, and the problem domain itself. The knowledge bases for many expert systems are, however, developed for a problem domain using only one domain expert, one knowledge engineer, and one elicitation method for a predetermined knowledge representation scheme

(or shell). The predictive accuracy of expert systems has to be tested for there is minimal (if any) research demonstrating that the above sources of variability do not significantly affect the quality of the knowledge base, and research in areas related to knowledge engineering that suggest they do.

For example, there is a long line of psychological research in the field of interviewing, which is analogous in many respects to knowledge engineering, that has demonstrated significant interviewer effects (e.g., see Forsythe and Buchanan, 1989). Research by Hammond et al. (1986; 1987) with 20 highway engineers using three knowledge elicitation methods to make aesthetics, safety, and capacity judgments found significant differences in predictive accuracy for experts, methods, problems and, most importantly, method-by-problem interactions. Research by Kahneman and Tversky (1984) clearly demonstrates that the way a problem is "framed" significantly affects people's decision making process. Research by Dawes and Corrigan (1974) and Hammond et al. (1975) has demonstrated that the inherent predictability of problem domains varies significantly. And, although they failed to vary knowledge engineers, the analysis of structured interviews obtained in a knowledge engineering context by Leddo and Cohen (1987) suggests that the amount and type of knowledge varies depending on the expert, task, knowledge representation scheme, and elicitation method; moreover, they suggest that interactions may exist among all of these sources of variation.

Given the many different threats to the predictive accuracy of the knowledge base, multiple experts and empirical evaluation methods should be used to evaluate the expert system's predictive accuracy. Experts, both those who participated in development and those acting as independent evaluators, are typically used to evaluate the predictive accuracy and, thus, the adequacy of the knowledge base. Expert evaluation typically proceeds in two ways: through examination of the knowledge base and the evaluation of test cases.

Examining the Knowledge Base. Expert examination of the knowledge base typically focuses on whether the system exhibits "correct reasoning."

The obvious concern is, of course, that the knowledge base not have mistakes. However, another concern—and one which Gaschnig et al. (1983) pointed out is not shared by all developers—is whether their expert systems reach decisions like human experts do. Many psychologists have long argued that this concern cannot be answered for one cannot, so to speak, look inside an expert's head to obtain the "correct reasoning." Instead, all one can do is build "paramorphic models" (Hoffman, 1960) of the reasoning process and evaluate their predictive accuracy against test cases. Indeed, researchers (e.g., Dawes and Corrigan 1974; Einhorn and Hogarth, 1975 ; Levi, 1989; Stewart et al., 1988) have shown that simple linear models can often result in prediction as good as those achieved by experts or the far more complex models found in expert systems.

As Lehner and Adelman (in press) point out in their review of the literature, this is not a resolved issue. On the one hand, Gaschnig et al. (p. 255) point out, "... there is an increasing realization that expert-level performance may require heightened attention to the mechanisms by which human experts actually solve the problems for which the expert systems are typically built." In addition, Adelman, Rook, and Lehner (1985) found that domain experts' judgments of the utility of decision support system and expert system prototypes were significantly affected by the match between how they and the system attempted to solve the problem. On the other hand, the intended users of many expert systems are novices, not experts. Research by Lehner and Zirk (1987) indicates that novices can perform extremely well with expert systems as long as they have a good mental model of how the expert system is using the data to arrive at its conclusions. What all of this suggests is that, at a minimum, the system's representation and presentation schemes need to be reviewed by experts and discussed with intended users.

Using Test Cases. We strongly advocate the use of test cases and performance standards to assess the predictive accuracy of the knowledge base. That is, we recommend an empirical test and evaluation of the knowledge base. We distinguish here between the empirical test and evaluation of the knowledge base and that of the entire expert system. Remember, the expert system may be addressing only part of a much larger organiza-

tional decision. Even if the technical test and evaluation of the knowledge base shows that it has good predictive accuracy, the expert system's contribution still may not ensure better organizational performance. Moreover, the potential users of expert system technology may not be experts in the substantive domain. In these cases, one needs both experts and users to participate in the evaluation. The experts are needed for the technical test and evaluation of the knowledge base; the users for the empirical test and evaluation of system performance. If possible and appropriate, experts should also participate in the empirical evaluation in order to systematically assess whether system performance is a function of user type.

There are two standards for empirically assessing the adequacy of an expert system's knowledge base. The first standard is ground truth; that is, comparing the system's predictions to the correct answers for the test cases. The second standard is expert judgment; that is, comparing the system's predictions to those of domain experts who have and have not participated in development. Ideally, one would like to be able to use both standards. Correct answers are desirable because substantial research (referenced above) has shown that experts do not always make perfect inferences and, in fact, often disagree with one another in the kinds of complex domains for which many expert systems are developed. Expert judgments are desirable because it is often inappropriate to expect better predictive accuracy from the system than from the expert. It is important to note, however, that this may not be the case where the system incorporates knowledge from a limited, well defined domain—such as a procedure manual—or where the system represents the expertise of several experts or is supporting a life and mission-critical mission. Under such conditions, it may be quite appropriate to expect the expert system to make more accurate predictions than any given expert.

When using test cases, we want to be able to perform statistical tests to assess the adequacy of the expert system's predictive accuracy. That is, we want to perform a t-test or binomial test or other appropriate statistical test—different tests will be considered in Chapters 5 and 6—to assess whether, on the average, (1) the expert system's predictions

are significantly different from the correct answers for the test cases, or (2) different from some predetermined performance level. Ideally, what we'd like to see is that, on the average, the system's knowledge base predicts the correct answer; that is, that there is no statistical basis for concluding that the expert system will not, on the average, predict the correct answer. This goal may, of course, be too ambitious for the expert system. As O'Keefe and O'Leary (1990) point out, the system may be acceptable if it performs at a predetermined level, such as being correct 90% or 95% of the time, or is as accurate as a graduate student if not an expert, or whatever the predetermined "acceptable performance range" is. [From a user and organizational perspective, we want performance to be as good, and preferably better, with than without the expert system.]

Ideally, we want to be able to conclude either that (a) the system's knowledge base is adequate (according to the predefined "acceptable performance range") when it is adequate, or (b) that it is inadequate when it is inadequate. It is important to note here that statistical tests are based on the probability calculus and, therefore, their conclusions are potentially fallible. In particular, there are two types of potential errors. The first type of error is called a Type I error. In the context of assessing the adequacy of the expert system's predictive accuracy, it is concluding that the knowledge base is not adequate when, in fact, it is. O'Keefe et al. (1987) refer to this as the "builder's risk." The second type of error, Type II error, would be concluding that the system's predictive accuracy is adequate when in fact it is not. O'Keefe et al. refer to this as the "user's risk." [These types of errors and, more generally, statistical testing will be considered in Chapters 5 and 6.]

More generally, if ground truth measures exist, one should try to discriminate between "accuracy" and "bias" in a signal detection sense (Lehner, 1989; Lehner and Ulvila, 1989). Accuracy refers to the degree of overlap in the distributions of belief values when the hypothesis is true versus false. Bias refers to the proportion of false negatives (hypothesis true, but user or system says false) to false positives (hypothesis false, but user or system says true). The two different types of bias are conceptually identical to Type I and II errors. This more general approach not

only considers "builder's risk" and "user's risk," but the expert system's ability to discriminate among alternative hypotheses. Moreover, it has considerable implications for helping the test and evaluation team decide on the number of test cases to use to evaluate the knowledge base's predictive accuracy (we elaborate on this in Chapter 5).

The level of bias to be accepted in an expert system is a critical decision for intended users and their sponsoring organizations. As the evaluation of DART (discussed in Chapter 2) demonstrated, different types of inferential errors differ in their implications, and thus importance, to decision makers. It is the evaluator's job to make sure that intended users and sponsors know the amount and proportion of different types of inferential errors to which the knowledge base is susceptible throughout development. For these reasons, Chapter 5 discusses the more general approach in detail.

If the correct answers do not exist or, for whatever reason, are inappropriate for the test cases, then one must rely on the judgment of an expert or the consensus judgments of a group of experts. Considerable care must be given to structuring the experts' activities. In particular, as with the DART test and evaluation, the evaluation team must ensure that the experts are "blind" as to whether the system or other experts generated the conclusions to the test cases. This is typically referred to as a "Turing test" (e.g, see Rushby, 1988; Yu et al., 1979).

In closing this section, it is important to note that test case construction is an important issue. To quote O'Keefe et al. (1987, p. 83), "The issue is not the *number* of test cases, it is the *coverage* of test cases—that is, how well they reflect the input domain. The input domain is the population of permissible input..." [italics theirs]. The required coverage capabilities is clearly a statement that needs to be a result of the requirements analysis. For as O'Keefe et al. point out, developers frequently devote a disproportionate amount of time to attempting to ensure that the system can handle the truly "expert" cases that may occur very infrequently. Moreover, these "infrequent" cases often become the test

cases. This may or may not be appropriate depending on the requirements for the system, and it can certainly be expensive.

An alternative identified by O'Keefe et al. is to randomly select test cases using a stratified sampling scheme such that the relative frequency of the cases is representative of those in the operational environment or stipulated in the requirements. Additionally, test cases should be chosen to cover situations where a failure in the system would be especially serious. It is also important that some of the test cases simulate the most common operation of the system. As Chapter 5 will show, a surprising small number of test cases are required to assess whether an expert system's level of predictive accuracy is sufficiently good to be of practical value to its users.

Summary. This section of Chapter 4 considered methods for evaluating the functional completeness and predictive accuracy of the knowledge base. Functional completeness is typically assessed by having experts (1) examine the knowledge base and question the developers on the various conditions the system can handle or not, and (2) use test cases to exercise the knowledge base. Predictive accuracy is assessed using test cases and comparing the expert system's performance against two standards: ground truth and expert judgment. Statistical tests are used to help decide whether the system's predictive accuracy is acceptable or not. In this regard, we discussed "builder's risk" and "user's risk," and the more general problem of assessing the types of errors (or bias) to which the system is most susceptible. These issues will be considered in substantial detail in Chapter 5. Finally, we addressed the issue of test case construction in closing this section.

INFERENCE ENGINE

The inference engine is the portion of the expert system that contains the general problem-solving knowledge. This includes an interpreter that decides how to apply rules to data and infer conclusions (or the analogous operations with other knowledge representations) and a schedule that controls the order in which the rules are applied. In many expert

systems, the inference engine is embedded in the development environment, tool, or shell. However, the vendors of such products provide little or no information on whether and how their inference engines were tested or validated.

As a practical matter, it is difficult to test an inference engine, and most testers do not even try. For noncritical applications of widely used and established environments, tools, or shells, this practice should not cause a serious problem. The widespread use of the tool will probably turn up most of the problems with the inference engine, and the noncritical nature of the application limits the seriousness of possible problems. Furthermore, other tests (e.g., those aimed at discovering the correctness of reasoning or of conclusions) can find some of the problems that could be caused by a faulty inference engine.

The development of benchmarks would aid in the testing of inference engines. A benchmark is a standard module of coded knowledge with known, proven-correct results that can be coded on a variety of inference engines. The correct performance of the inference engine on a comprehensive set of benchmarks provides strong evidence that the inference engine is correct. Unfortunately, such a set of benchmarks has not yet been developed.

Another approach to testing inference engines is to code identical knowledge bases with different inference-engine products (e.g., shells) and exercise these programs in parallel. If the behavior of all of the systems is the same, this gives some evidence that the inference engines are free of problems. However, this evidence is not absolutely conclusive. Furthermore, different results indicate problems in one or more inference engines, but may not indicate which particular inference engine is faulty. This procedure is also expensive, and the expense will be hard to justify for any but the most critical expert systems.

SERVICE REQUIREMENTS

In addition to evaluating the expert system's knowledge base and inference engine, traditional software test and verification methods can be

used to help assess what Rushby (1988, p. 75) has called the expert system's "service" versus "competency" requirements. These methods have considerable applicability (a) prior to programming code for verifying requirements analysis documentation and functional models of the software, and (b) once the development process is well underway, during hardware configuration, system packaging, and system transfer.

Verification testing should be systematically performed for the service requirements of expert systems, just like any other software product. Fagan and Miller (as reported in DeMillo et al., 1987) have identified four phases for software testing. The first phase is manual analysis in which the requirements specification, the design and implementation plan, and the program itself are analyzed for problems by experienced software engineers. The second phase is static analysis, which may be manual or automated, in which requirements and design documents and software are analyzed, but without code execution. The third phase is dynamic analysis in which software is executed with a set of test data. And the fourth phase, which Fagan and Miller consider to be optional, is attempting to prove the program as being correct. [This last phase assumes a stable program, not a prototype, and is typically reserved for critical modules.]

According to classification found in DeMillo et al. (1987), most testing techniques seem to focus on static or dynamic analysis; that is, the second and third phases in Fagan and Miller's taxonomy. Moreover, the first and fourth phases can be subsumed by the static and dynamic phases, respectively, to simplify the presentation. This will make the focus on testing "service" requirements comparable to the focus on using static and dynamic testing methods to test characteristics of the knowledge base considered earlier in this chapter. Consequently, software testing methods will be overviewed under these two category headings below.

DeMillo et al. (1987) list five static analysis methods. The first four methods are manual; the last one is automated. The manual static analysis methods are (1) requirements analyses, (2) design analyses, (3) code inspections and (4) walkthroughs. Requirements analyses typically use

a checklist of evaluation criteria, such as the consistency among the different requirements specifications of the system, their necessity in achieving system goals, and their implementation feasibility with existing resources. Design analyses also use a checklist, which may actually be quite similar to that used in the requirements analysis, but now the focus is on elements of the software system design, such as the data flow diagrams, module interfaces, algorithms, etc. Code inspections and walkthroughs involve the static analysis of the program by a group of people but, according to DeMillo et al., the former uses a checklist of common programming errors as a reference point and the latter uses a set of test cases for assessing the logic of the program. In addition, as Fairley (1985) points out, inspections differ from walkthroughs in that a team of trained inspectors analyze the work products in the former.

Static analyzers, which is what the term "static analysis" typically connotes, are automated tools that analyze the source code for logic errors, structural errors, syntactic errors, coding style, interface consistency, etc. without using test cases to execute the code. As with automated static analysis of the knowledge base, traditional software static analyzers are valuable but they do suffer from both practical and theoretical limitations.

In dynamic testing, test data (called cases) are constructed and used to execute the software in an effort to uncover programming errors. Fairley (1985) identifies four types of tests: (1) functional tests, (2) performance tests, (3) stress tests, and (4) structural tests. Functional tests are designed to evaluate the adequacy of the software in performing the functions identified in the requirements specifications. Test data are selected by specifying typical operating conditions and input values, and examining whether or not they result in the expected outcomes. Performance tests are also tied to the requirements (and design) specifications, but now the focus is on, for example, verifying the response time under various loads, determining the amount of execution time spent in various parts of the program, and examining program throughput. Stress tests are, as the name suggests, designed to overload and, in many cases, break the system in an effort to assess its strengths and limitations. Finally, structural

tests are designed to exercise the logic of the program by traversing various execution paths. These types of dynamic tests are just as appropriate for assessing the quality of the system's "competency," as its "service" requirements.

There are a number of different dynamic testing methods. Three methods are introduced here; discussion of these and other methods can be found in numerous texts, including DeMillo et al. (1987), Fairley (1985), Pressman (1982), and Rushby (1988). The three methods are (1) random testing, (2) input space partitioning, and (3) symbolic testing. Random testing is a strategy in which a program is tested by randomly selecting a subset of all possible input values. The distribution of input values can be either arbitrary or attempt to reflect the distribution actually found (or expected) in the application environment. In input space partitioning, test data are selected for evaluating the different subsets of the prog am input domain, such that each partition causes the execution of a different program control path. The concept is that the input space is partitioned (or divided) into groups such that the inputs within each group are in some sense equivalent and, therefore, likely to result in similar behavior. Again, both random testing and input space partitioning methods can also be used to test the knowledge base.

In contrast to random testing, input space partitioning, and most testing methods, symbolic testing uses symbolic inputs (e.g., symbolic constants) and outputs (e.g., symbolic formulae and symbolic predicates) to evaluate program accuracy. The underlying assumption of symbolic testing is that a program can be conceived of as a finite set of assertion-to-assertion paths that can be represented symbolically by an execution tree consisting of nodes associated with the statements being executed and directed arcs indicating program flow. The objective is to demonstrate symbolically that each assertion path is accurate and, in turn, that the program is correct. In this regard, symbolic testing is comparable to the theorem-proving methods considered above for testing the knowledge base.

Software verification testing methods are essential if the expert system is embedded in other software, for having a high-quality expert

system embedded in low-quality, deficient software will be of little value to the user. However, in many cases, the expert system will not be embedded in other software, but will have to be compatible with the computer system in the user's operational environment. As Cholawsky (1988, p. 44) points out, "In general, prototypes ignore both deployment issues (such as cost-benefit analysis, scaling up to operational size, and handling real-world data) and transition issues. Important problems, such as linking to standard databases and porting to standard user hardware, may not be investigated by the prototype."

"Deployment" and "transition" issues are important ones for testers to consider. In an effort to assist them, we have identified a host of such issues and listed them as attributes under the "service requirements" branch of the MAU hierarchy in Table 2-1. In particular, these attributes are classified into three groups: those dealing with (1) the computer system, (2) computer usage, and (3) system integration. The individual attributes in these three groups are defined, in turn.

- Computer System:
  - *Design*. The extent to which the expert system runs on the approved computer hardware and operating system and utilizes the preferred complement of equipment and features. In some cases, the original system requirements may specify or describe the preferred or required system. In other cases, the tester may need to survey available equipment at the intended installation.
  - *Portability*. How easily the expert system can be transferred to other computer systems.
- Computer Usage:
  - *Set-Up Time*. The amount of time required for the computer operator to locate and load the program (if any) and the time to activate the program. Set-up time should be measured in the expected operating environment (i.e., how the program will actually be implemented.)
  - *Run Time*. The amount of time required to run the program with a realistic set of input data. This factor refers only to the time that the computer program takes to run; the time needed for the programmer and user is included under dynamic testing factors.

- *Space Requirements.* The amount of RAM and disk space required by the program.

- *Reliability (Hardware).* Percentage of time the computer system could be expected to be operating effectively.

- *Capability (Hardware).* The computer system's total amount of RAM and disk space.

- *Effect of Feature Use/Jumping.* The extent to which moving from various parts of the program causes errors.

- *Degradation.* How well the program (a) saves data and analyses, and (b) permits continuation after an unexpected program or system crash or power outage.

- *Handling Input Errors.* The extent to which the program (a) prohibits a program crash, and (b) tells the user what to do after an input mistake.

• System Integration:

  - *Formats.* The extent to which the program uses input and output formats that are consistent with the intended use. This includes any mandated or standard formats that are specific to the intended user organization.

  - *Data Requirements.* The extent to which the program's data requirements are consistent in content, quantity, quality, and timeliness with those available to the intended user organization. The program should also be able to interact with specified and appropriate databases and communications systems.

  - *Documentation.* The adequacy of material regarding the program's use and maintenance. User's manuals should be complete and understandable. Copies of computer code and its supporting documentation should be complete and understandable, and should allow maintenance by the government. (All applicable DoD software documentation standards should be met.)

  - *Skill Requirements.* The extent to which the program can be operated by appropriately skilled individuals. The appropriate skill requirement includes grade level (for military enlisted, military officer, or civilian personnel), users' technical background, and training requirements. The appropriate level may be specified in requirements or may be determined by reference to the organizational setting of its intended use and to the personnel assigned to that setting.

This section of the chapter has overviewed traditional software test and verification methods for assessing the adequacy of the expert system's "service requirements." In particular, we have overviewed five traditional static testing methods and four dynamic testing methods. These methods are most applicable when the expert system is embedded in a larger software system. When it is not embedded in a larger system, we have defined a number of attributes for assessing the expert system's compatibility with the computer system in the user's operational environment.

CHAPTER SUMMARY

Chapter 4 addressed technical test and evaluation methods for assessing how well the expert system is built. Its three sections were organized around the first three criteria in the MAU hierarchy in Table 2-1. In particular, the first section dealt with testing and evaluating the expert system's knowledge base. The second section dealt with testing and evaluating the inference engine, and the third section, the expert system's service requirements. Three classes of technical test and evaluation methods were considered: (a) static and dynamic testing methods for assessing the logical consistency and completeness of the knowledge base and the adequacy of the inference engine; (b) methods for using domain experts to assess the functional completeness and predictive accuracy of the knowledge base; and (c) conventional software test and verification methods for assessing the service requirements of the entire system.

CHAPTER 5:

MORE ON ASSESSING THE PREDICTIVE ACCURACY OF AN EXPERT SYSTEM'S KNOWLEDGE BASE

Because of its importance, Chapter 5 extends the discussion in Chapter 4 on approaches to testing and evaluating the predictive accuracy of the expert system's knowledge base. For simplicity, we use the terms "performance of the expert system" or "performance evaluation" to refer to "assessing the predictive accuracy of the expert system's knowledge base." The reader should not confuse the discussion in this chapter with the discussion in the next, which is oriented to assessing the overall performance of the person and organization using the expert system. However, both chapters emphasize the use of empirical concepts and methods.

The approach one takes to performance evaluation depends on the objectives for which the expert system was developed. If the objective of the system is to capture and encode human expert problem solving, then performance should be evaluated vis. a vis. similarity to expert judgment. Alternatively, if the system objective is to maximize accuracy, then performance evaluation must estimate the extent to which the system generates correct or accurate outputs.

Below we consider several possible evaluation procedures, each corresponding to a different development objective. [This chapter contains highly technical material on a small portion of the hierarchy—performance against ground truth. This chapter is not essential to the understanding of the other chapters.]

CASE 1: HYPOTHESIS TESTING WITH BELIEF VALUES

The first case we will consider is presented in more detail (and conceptually may be more difficult) than some of the cases presented below. However, most of the concepts we use in the later cases can be conveniently introduced here.

Consider the case of an expert system that outputs quantitative belief values for alternative hypotheses. The objective of the system is to assess the relative likelihood (plausibility, belief, etc.) of each of a set of hypotheses. As we saw in earlier chapters, many systems fall into this category.

Possible Performance Measures

As discussed in Lehner (1989) and Lehner and Ulvila (in press), an expert system that evaluates predefined hypotheses is loosely analogous to a signal detector. A signal detector is any system that functions to discriminate occurrences from nonoccurrences of a signal. As shown in Figure 5-1, the signal detection problem is often characterized as one of receiving a set of sample values (perceived signal strength) from one of two distributions (signal exists vs. signal does not exist), and on the basis of this information, deciding from which of the two distributions the signals were drawn. Usually this decision is based on whether the observed signal strengths exceed a threshold. The decision threshold is determined from background knowledge of the underlying distributions. The sensitivity of a signal detector is often measured in terms of the normalized difference between the means of the two distributions (d'). If d' is large (small), then the error rate of signal/no signal decisions will be small (large).

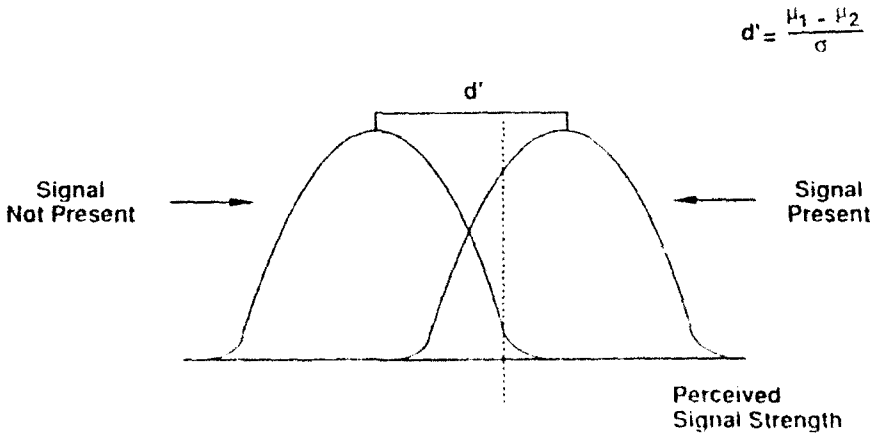


Figure 5-1: Hypothetical Distribution of Perceived Signal Strength in Signal Detection Theory

In an expert system inference network (or other knowledge structure), each node represents two or more mutually exclusive hypotheses. Many expert systems generate a belief value (probability, certainty level, Shaferian belief, etc.) for each hypothesis. Consider a node that discriminates two hypotheses, H_1 and H_2. When H_1 is true, we would generally expect the belief value in H_1, bel(H_1), to be higher than when H_2 is true. The user's problem is to use the belief values output by the expert system, along with other available information, to select a hypothesis and act accordingly. If there is a large (small) difference between the mean bel(H_1) value when H_1 vs. H_2 is true, then the expert system should be useful (useless) in helping a user to discriminate these two hypotheses.

From a user perspective, an expert system is useful if it helps the user discriminate instances when different hypotheses are true. One approach to evaluating a system is to estimate the proportion of times the expert system will generate advice that is useful in discriminating among alternative hypotheses. In this section we show how this can be done. In this section and the next section (Cases 1 and 2), we will make several assumptions. Each assumption will be discussed or relaxed in Case 3.

Assume an expert system that distinguishes between just two hypotheses, H and ~H. Assume also that the expert system generates belief values that satisfy bel(H) = 1-bel(~H). (Call these assumptions A1 and A2, respectively.) Consider Figure 5-2, which contains two distributions [or densities], P(bel(H)|H) and P(bel(H)|~H). Two thresholds have been set, U and L. Depending on how a user utilizes an expert system, Figure 5-2 has at least two different interpretations.

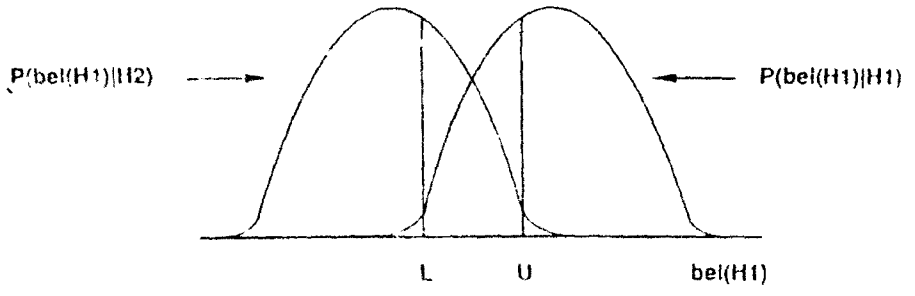


Figure 5-2: Distributions of Belief Value, bel(H1) for H1-true vs. H2-false

First, the expert system may be utilized to partially automate the inference process. That is, if the expert system outputs very high (low) belief values, then the user simply acts under the assumption that H (~H) is true. In this context, U and L can be interpreted as decision thresholds. If bel(H) is greater (less) than U (L), then the user concludes H (~H). Otherwise the user is uncertain, and proceeds to collect additional evidence. Of course, complete automation occurs when U = L.

Alternatively, the user may view the expert system as a source of evidence. That is, the user combines the expert system's output with other data and knowledge to make his or her own inferences. In this context, an important question to ask is "How often does the expert system output strong evidence for the correct conclusion?" One standard approach to measuring the strength or diagnosticity of an item of evidence is by a likelihood ratio:

$$LR = \frac{P(bel(H)|H)}{P(bel(H)|\sim H)}.$$

If LR is high (e.g., greater than 10), then the report "bel(H)" is strong evidence for H vs. ~H. If LR is low (e.g., less than .1), then the report "bel(H)" is strong evidence for ~H.

The reason that LR is a standard measure of evidential value is that most theories of rational induction (i.e., proper degrees of belief) recommend the use of Bayes' Rule for updating (see Mortimer, 1988). This rule states that a person's relative degree of belief in H vs. ~H, given a new piece of evidence, E, should be determined by

$$\frac{P(H|E)}{P(\sim H|E)} = \frac{P(E|H)}{P(E|\sim H)} * \frac{P(H)}{P(\sim H)},$$

$$\text{Posterior Odds} = \quad LR \quad * \text{ Prior Odds}.$$

U and L in Figure 5-2 can be interpreted as thresholds of strong evidence. That is, if bel(H) is greater (less) than U (L), then the expert system has output strong evidence for (against) H. If bel(H) is between U

and L, then that output does not provide strong evidence in either direction; that is, the user will need to base his or her decision on other factors or be driven by priors.

Consequently, whether the user chooses to utilize the expert system to partially automate inference decisions or as a source of evidence, Figure 5-2 provides a way of characterizing user/expert system interactions.

Consider P(bel(H)|~H), the distribution of belief values when H is false. Given U and L, we can specify three probabilities:

P(bel(H)<L|~H) ← probability of true negative

P(U>bel(H)>L|~H) ← probability of uncertain output

P(bel(H)>U|~H) ← probability of false positive.

Similarly,

P(bel(H)<L|H) ← probability of false negative

P(U>bel(H)>L|H) ← probability of uncertain output

P(bel(H)>U|H) ← probability of true positive.

Define P_F to be the probability that the expert system will generate belief values that strongly support the wrong conclusion and P_U to be the probability that the expert system will generate belief values that do not provide strong support for either conclusion. From the above six probabilities, we know that

$$P_E = P(bel(H)<L|H)*P(H) + P(bel(H)>U|\sim H)*(1-P(H)) \qquad [5\text{-}1]$$

and

$$P_U = P(U>bel(H)>L|H)*P(H) + P(U>bel(H)>L|\sim H)*(1-P(H)) \qquad [5\text{-}2]$$

where P(H) is the probability (anticipated relative frequency) of sampling from the H-true distribution.

Together, P_E and P_U are two aggregate measures of the usefulness of an expert system. If P_E is relatively high, say .1, then the expert system is generating outputs that strongly support the wrong conclusion about 10% of the time. If P_U is relatively high, say .3, then the expert system is generating useless outputs approximately 30% of the time. From these two numbers, we know that $1-P_U-P_E$ is a measure of the proportion of times the expert system will strongly support the correct conclusion.

Estimating P_E and P_U

One of the objectives of evaluating an expert system is to assess the extent to which that expert system can help a user to make correct inferences. Although different users will set different U and L thresholds, one can still ask whether it is possible to set thresholds where P_E and P_U are *simultaneously* low. If this cannot be done, then the expert system cannot be very useful in as much as the user must either tolerate a high error rate or a high rate of outputs in the uncertain region.

To estimate the extent to which P_E and P_U can be simultaneously low, it is useful to make several simplifying assumptions. They are as follows.

A3) Given each hypothesis, the distributions of belief values are normally distributed.

A4) The distributions of belief values have equal variance.

A5) The U and L thresholds are symmetric. This means that

$$(bel(H)>U|\neg H) = P(bel(H)<L|H).$$

Since the thresholds are symmetric and the two normal distributions have equal variance, it follows that P_E and P_U are now independent of the relative frequency of sampling from each distribution. Or equivalently, P_E and P_U are not affected by the prior probability P(H). Specifically, we get

$$P_E = P(bel(H)>U|\sim H) = P(bel(H)<L|H),$$

and

$$P_U = P(U>bel(H)>L|\sim H) = P(U>bel(H)>L|H).$$

In addition, from these assumptions it follows that

$$(M_1-M_2)/s = z(1-P_E) + z(1-P_E-P_U), \qquad [5\text{-}3]$$

where M_1, M_2, and s are the means and standard deviation of the two distributions, and z(X) is the z-score for X. From this it can be seen that any procedure for estimating the means and standard deviation of the two distributions will also provide an estimate of P_E and P_U.

Consequently, one can specify a straightforward test procedure for evaluating an expert system that discriminates H and ~H. First, identify two representative sources (H-true vs. H-false) of possible test problems. Randomly select problems from each source. Run the expert system against each problem and do a t-test comparison of the results. The t-test analysis will output an estimate of the mean and standard deviation of each distribution, an estimate of the difference between the means of the two distributions, and a standard error of the estimate for this difference. From these three estimates, P_E and P_U can be estimated in turn using Equation [5-3]. An example of this will be given shortly.

Using P_E and P_U to Determine Sample Size

Although the above procedure is straightforward, we still need to determine the number of test problems required. As it turns out, the P_E and P_U measures can be helpful in making this determination. A standard result from classical statistics (see Hays, 1972, p. 417-422) will be useful here. Namely,

$$N = \frac{2[z(1-\alpha) - z(\beta)]^2}{(M_1-M_2)/s}$$

where N estimates the number of test problems per condition needed to guarantee that if the difference between the two distributions is at least (M_1-M_2/s), then there is at least a 1-probability of obtaining significance at the α level in a one-tailed t-test of the null hypothesis of no difference.

Using this equation, we can determine a minimum sample size for both groups by specifying the following parameters:

max P_E - a maximum acceptable error rate,

max P_U - a maximum acceptable rate of ambiguous results,

α - significance level for t-test

$(1-\beta)$ - the power of the t-test.

Given these numbers, the minimum sample size for each group is derived as follows:

$$N = \frac{2[z(1-\alpha) - z(\beta)]^2}{[z(1-\max P_E) + z(1-\max P_E-\max P_U)]^2}.$$

If $P_E + P_U \leq \max P_E + \max P_U$, then the probability of obtaining a statistically significant difference (at the α level) between the two groups is at least $(1-\beta)$. As will be illustrated below, using this equation will often result in a minimum sample size that is very small (around 5 tests for each hypothesis in a node).

An Example

Assume that we have been given the responsibility of testing an expert system with the simple inference network shown in Figure 5-3. In this inference network, there are three evidence items (evid1, evid2, and evid3), one intermediate hypothesis (ihyp1), and one goal hypothesis (ghyp1). Although the analysis does not depend on how belief values are calculated, we note here that bel(ihyp1) is a linear function of bel(evid2)

and bel(evid3), and bel(ghyp1) is calculated by performing a relative maximum entropy update given new values for bel(evid1) and bel(ihyp1).

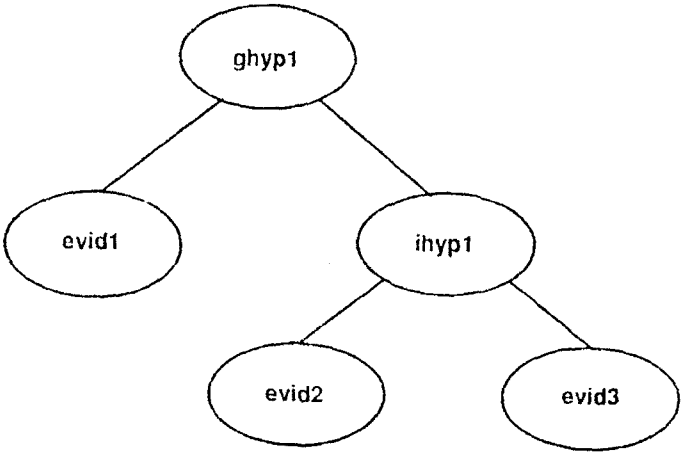


Figure 5-3: Sample Inference Network

Our first task is to specify a minimum sample size. As evaluators we make the following judgments:

(1) For both false positives and false negatives, an error rate greater than 5% is unacceptable. If the error rate is larger than this, users will simply discard the system.

(2) The system should not generate ambiguous results more than 30% of the time. Beyond this level, using the system will be perceived as more trouble than it's worth.

(3) Set $\alpha = .05$---a level commonly used.

(4) Set $(1-\beta) = .90$. If, indeed, the system satisfies (1) and (2) above, then the probability of obtaining one-tailed t-test significance at $p \leq .05$ is .9 or greater.

From these four judgments, we get

$$\max P_U = .30,\ \max P_E = .05,\ \alpha = .05,\ \text{and}\ (1-\beta) = .9.$$

This gives us

$$N = \frac{2[z(1-\alpha) - (\beta)]^2}{[z(1-P_E) + z(1-P_E-P_U)]^2} = \frac{2(1.65 - (-1.28))^2}{[1.65 + .39]^2} \approx 4.12.$$

So the minimum sample size is approximately four test problems per condition. Even though this seems like a small sample size, if the difference between the two distributions is substantial (i.e., difference between means sufficient to give $P_E + P_U \leq .35$), then there is a 90% chance that this small experiment will generate a t-test result with $p \leq .05$. Consequently, it is unlikely that the expert system, if it satisfies these criteria, will not exhibit at least some difference between the two distributions.

We decide to be "conservative" and let N = 8.

After running the 16 randomly selected tests, we get the results shown in Table 5-1. A standard t-test applied to ghyp1 indicates a statistically significant difference between the two sample distributions ($p \leq .00005$). Clearly the expert system has achieved some discrimination between H and ~H.

Table 5-1: Sample Test Results

Group	Output Belief Values				
(gyp1-true = 1)	evid1	evid2	evid3	ihyp1	ghyp1
0	.4	.56	.61	.59	.39
0	.4	.43	.28	.36	.33
0	.33	.78	.29	.54	.3
0	.26	.23	.33	.28	.28
0	.48	.26	.32	.29	.34
0	.24	.34	.36	.35	.29
0	.29	.54	.78	.66	.38
0	.48	.48	.34	.41	.37
1	.69	.21	.89	.55	.45
1	.8	.76	.56	.66	.5
1	.44	.89	.48	.69	.42
1	.61	.76	.94	.85	.5
1	.81	.55	.86	.71	.52
1	.59	.56	.4	.48	.4
1	.76	.48	.69	.59	.47
1	.68	.49	.23	.36	.4

In addition, we estimate a minimum value for P_U by

$$z(1\text{-max } P_E) + z(1\text{-max } P_E\text{-est}[P_U]) = \text{est}[M_1\text{-}M_2)/s].$$

The observed mean difference is .119, and the estimate of the standard deviation of the distributions is .044. This gives us

$$z(.95) + z(.95\text{-est}[P_U]) = \text{est}[M_1\text{-}M_2)/s]$$

$$1.65 + z(.95\text{-est}[P_U]) = .119/.044$$

$$z(.95\text{-est}[P_U]) = 1.05$$

$$.95\text{-est}[P_U] = .85$$

$$\text{est}[P_U] = .1.$$

This procedure can be repeated for alternative levels of max P_E, from which one can see the tradeoff between P_E and P_U. This is illustrated in Table 5-2.

Table 5-2: Tradeoff between P_U and P_E in Sample Problem

max P_E	est P_U
.1	-.02*
.05	.1
.025	.21
.01	.35
.005	.45
.001	.56

* indicates distributions are sufficiently separated that a single threshold can be set where

$$P(\text{bel}(H)>L|\sim H) = P(\text{bel}(H)<U|H) \leq \text{max } P_E.$$

Finally, a 90% confidence level for the minimum value of P_U can be estimated by (a) calculating the 90% confidence level for the minimum mean difference and (b) repeating the above procedure. In the case of ghypl, the observed difference was .119 and the standard error of the estimate of the difference was .022. Consequently, the 90% confidence level for the

difference is .119 - .022*t(.9, df=14), which is .09. This gives us an "upper bound" on P_U of

$$1.65 + z(.95\text{-}est[P_U]) = .09/.044$$

$$z(.95\text{-}est[P_U]) = .40$$

$$.95\text{-}est[P_U] = .66$$

$$est[P_U] = .29.$$

A similar analysis can be performed for all the nodes in the network. The t-test results for each node in the sample problem are summarized in Table 5-3.

From Table 5-3 we can draw several conclusions. Overall, the expert system performs well. As far as the goal node (H vs. ~H) is concerned, a user willing to tolerate a 5% error rate should find the expert system advice useful more than 71% of the time, and most likely around 90% of the time. These results do support the evaluation hypothesis that $P_U \leq .3$.

Table 5-3: Test Results for all Nodes in Sample Problem (max P_E set at .05)

Node	M_1-M_2	Estimate Standard Deviation	Estimate Standard Error	(max P_E = .05) est P_U	90% C.L. est P_U
evid1	.32	.116	.058	.08	.28
evid2	.135	.197	.099	.78	.90
evid3	.218	.222	.111	.69	.86
ihyp1	.176	.147	.073	.63	.82
ghyp1	.119	.044	.022	.1	.29

Regarding the other nodes in the network, it seems that most of the discrimination is obtained from evid1, and that the other nodes contribute relatively little to the overall accuracy of the system.

Reconsidering the Assumptions

In Cases 1 and 2 several assumptions were made. They were:

A1) The expert system considers only two hypotheses, H and ~H.

A2) The belief values sum to one.

A3) The distributions of belief values are normal.

A4) The distributions of belief values have equal variance.

A5) The thresholds are symmetric.

Each of these assumptions is discussed below. Assumption A3 will be considered last.

Multiple Hypotheses. If there are just two hypotheses (H and ~H), with belief values that sum to one (A2), then bel(H) completely summarizes both values. When there are more than two hypotheses, this is no longer true. Given a belief value for one hypothesis, the belief values for the other hypotheses can still vary. This implies that for each hypothesis, there is a multivariate distribution of belief values. For instance, if the expert system discriminates three hypotheses, H_1, H_2, and H_3, then the output can be characterized as a vector of belief values:

$$b = \langle bel(H_1), bel(H_2), bel(H_3) \rangle.$$

There are two ways to address the multiple hypothesis case. The first is to perform a multivariate statistical analysis. The thresholds then become hyperplanes in a vector space of possible belief values. For instance, one might set thresholds U_i where for each H_i, the decision rule is to select H_i if $bel(H_i) > U_i$. The area defined by $bel(H_i) < U_i$ for all i would then be the uncertain region. P_U is the probability of falling into the uncertain region, while P_E is the probability that for some i, $bel(H_i) > U_i$ occurs when H_i is false. Conceivably one could generalize the evaluation procedure described in Case 2 to address this multivariate problem. We have not explored the details of this generalization.

An alternative approach is to do a pairwise comparison of hypotheses. This can proceed as follows. First, define a measure, Bel_{ij}, that summarizes the relative belief values of the two hypotheses. For example, we could set

$$bel_{ij} = bel(H_i)/[bel(H_i)+bel(H_j)],$$

or possibly,

$$bel_{ij} = [bel(H_i)-bel(H_j)].$$

Second, determine the minimum sample size required for each pairwise comparison. Third, select a sample size for each H_i that is greater than the maximum of the minimum sample sizes required for each pair comparison involving H_i. Finally, collect the test data and compare each pair of hypotheses as discussed above. This procedure provides a series of tests that are individually appropriate, but not statistically independent.

Belief Values that do not Sum to One. Many expert systems employ an uncertainty calculus where belief values do not sum to one or where a range of possible values is maintained for each hypothesis. For example, in a Shaferian system of beliefs (Shafer, 1976), bel(H) is often interpreted as the degree to which the existing evidence supports H, where it often occurs that $bel(H) + bel(\neg H) < 1$.

Conceptually this case is similar to the multiple hypothesis case. For each hypothesis, there is a multivariate distribution of belief values. Consequently, the same techniques apply here. In the case of Shaferian beliefs, for instance, it seems natural that for each pair of hypotheses H_i and H_j, $[bel(H_i) - bel(H_j)]$ effectively summarizes the extent to which the expert system finds evidence that supports H_i vs. H_j.

Unequal Variance. Assumption A4, that the two distributions have equal variance, is not essential. The main implication of violating this assumption is that P_U, but not P_E, now depends on the relative frequency of

sampling from the two distributions. This can be seen from Equations [5-1] and [5-2].

If assumption A4 is not made, then the procedure described in Case 1 needs to be modified to (1) estimate the variance of each distribution of belief values separately, and (2) incorporate an estimate of the relative frequency of sampling from each distribution. As long as the thresholds are symmetric, P_E is unaffected by unequal variances. However, P_U will vary, although its value is bounded by $P(U>bel_{ij}>L|H_i)$ and $P(U>bel_{ij}>L|H_j)$, where H_i and H_j are the two hypotheses being compared. A "conservative" estimate for (2) is one that pushes the value for P_U close to its maximum value.

Nonsymmetric Thresholds. For testing purposes, the assumption of symmetric thresholds is reasonable. Suppose a test that assumes symmetry (with $P_E = x$) yields an unacceptably high value for P_U. This would imply that for a decision maker to reduce the value of P_U, the decision maker must accept either P(false positive) > x or P(false negative) > x or both. Or equivalently, for a given P_U, the minmax value for P_E occurs when the thresholds are symmetric.

However, if the evaluator wishes to assume nonsymmetric thresholds, then both P_U and P_E will depend on the relative frequency of sampling from the two distributions. Consequently, if assumption A5 is violated, then the procedure in Case 2 must be modified to incorporate a subjective estimate of the relative frequency of sampling from each distribution. Note again that P_U is bounded by $P(U>bel_{ij}>L|H_j)$ and $P(U>bel_{ij}>L|H_j)$, while P_E is bounded by $P(bel_{ij}<L|H_i)$ and $P(bel_{ij}>U|H_j)$. Consequently, a "conservative" estimate of the relative frequency of sampling from the H-true distribution is an estimate that pushes $P_U + P_E$ towards its maximum value.

Distributions are Normal. Assumption A3 is expedient. Although normal distributions are prevalent in nature, there is no guarantee that belief values are always distributed normally. Furthermore, there are procedures for testing the hypothesis that a collection of sample points was generated from a normal distribution. When the test data suggest that

the distribution is not normal, then one should consider alternative procedures (see below).

It should be noted, however, that testing an expert system is often an expensive proposition. As a result, the sample size for each distribution is often small (less than ten). Given a small sample size, it is unlikely that a sample distribution will lead to rejecting the assumption of normality, even when the true distribution is not normal. When the normality assumption is incorrect, we are unlikely to detect it.

This leaves us with a quandary—routinely use weaker procedures that make fewer assumptions (viz., nonparametric statistics), or simply assume normality and accept the occasional errors in evaluation that this assumption will entail. In general, we prefer the latter option.

Some Non-Parametric Procedures

Even if normality is rejected, there are some procedures that can be used which are analogous to, albeit much weaker than, the parametric procedures discussed above. Specifically, we define a new measure, d*. For mutually exclusive hypotheses H_1 and H_2, let P1 and P2 represent the probability distributions P(bel(H1)|H1) and P(bel(H2)|H1), respectively. We define

$$d^* = P(x<y | x \in P1 \text{ and } y \in P2).$$

The d* statistic is a measure of the extent to which two distributions can be separated. From the perspective of decision thresholds, d* has the following properties:

Property 5-1: Given any P1 and P2 distributions, if P(H) = .5, then the probability of an error is at least d*/2.

Property 5-2: Given any P1 and P2 distributions, if the decision threshold is symmetric, then the probability of an error is at least d*/2.

From the d* statistic, therefore, we can learn something about the potential accuracy obtainable using a threshold decision rule. If d*/2 is high, then the expert system cannot be very helpful in discriminating H1 from H2.

Estimating d*

The empirical procedure for estimating d* is similar to the parametric case. Identify two representative sources (H-true vs. H-false) of possible test problems. Randomly select an equal number of problems from each source. Run the expert system against each problem and estimate d* as follows. Let S1 and S2 be the sample distributions corresponding to P1 and P2, respectively. If S1 and S2 both contain N observations, then there are N^2 pairs of sample points—one from each sample distribution. Let r be the number of pair reversals—each pair (x,y) where x S1, y S2 and xy. Let e be the number of equal pairs (if e is odd, add 1 to e). We then get

$$\text{estimate } d^* = \frac{(r + e/2)}{N^2}.$$

This will be a slightly conservative (over) estimate of d*.

To estimate a confidence level for d*, set n to Nd* and then round n up to an integer value. Then proceed to treat n as though it were n hits in a series of N Bernoulli trials. Using a binomial distribution, calculate a confidence level for p. This will be a conservative confidence level for d*.

To illustrate, consider Figure 5-4.

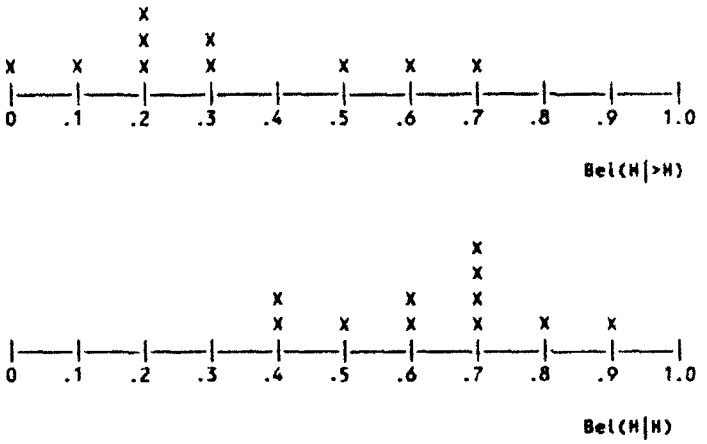


Figure 5-4: Sample Data for Calculating d*

Here N = 10, r = 10, e = 7. This gives us

$$\text{estimate } d^* = \frac{(10 + 4)}{100} = .14.$$

Furthermore, rounding up Nd^*, we get $n = 2$. In the binomial distribution at $p = .45$, the probability of getting 2 or less hits is less than .1. Consequently, at the 90% confidence level, we can assert that $d^* \leq .45$.

CASE 2: HYPOTHESIS TESTING WITHOUT BELIEF VALUES

Many expert systems are categorical. They simply output a recommended solution, often by testing preconditions for a predefined hypothesis. A standard technique (Green and Swets, 1966) for evaluating this type of system is to apply it to a set of test problems and fill in the following 2×2 matrix.

		Hypothesis is True	Hypothesis is False
Hypothesis Asserted to be	True	True Positive	False Positive
	False	False Negative	True Negative

From this matrix one can obviously estimate the probability of each type of error. Furthermore, one can also estimate the overall "sensitivity" of the expert system in discriminating true vs. false instances of the hypothesis. This is done by assuming that the process underlying the categorical assessments was a signal detection process with a single decision threshold with two identical normal distributions. This model was depicted in Figure 5-1. The normalized difference between the means of the two distributions defines the "sensitivity" of the signal detector. This statistic, called d', is simply calculated as

$$d' = [z(1\text{-}P(\text{false positive}) - z(1\text{-}P(\text{true positive})].$$

A small value for d' (e.g., $d' < 1.0$) would indicate that the expert system is not very sensitive to the hypothesis true vs. false condition, while a large value (e g., $d' > 2.0$) suggests that the expert system can effectively distinguish the two states.

This approach can be generalized to a case where the expert system also may generate an Unknown response. When this occurs, we can add a third "Unknown" row to the above matrix. Figure 5-2 becomes the underlying model for this matrix. The procedure for calculating d' remains the same.

Finally, it should be noted that is often used to measure the performance of a user/decision aid combination. That is, independent of the actual outputs generated by the expert system, one can test to see whether using the expert system increases a user's d' score.

Sometimes it is possible to assign utilities or costs to different types of outcomes. For instance, in medical diagnosis, the cost of a false positive may be only that of performing a second, more precise test, while the cost of a missed positive may be serious health consequences. If one can combine such costs and utilities into a single scale (call this the Value scale), then the expected value associated with using an expert system is simply

EV = P(true positive) * Value(true positive)

+ P(true negative) * Value(true negative)

+ P(false positive) * Value(false positive)

+ P(false negative) * Value(false negative).

Examples of this approach can be found in Levi (1985), Kalagnanam and Henrion (1988) and Heckerman (1987).

CASE 3: ASSESSING THE ACCURACY OF QUANTITATIVE PREDICTIONS

Many expert systems generate quantitative predictions as outputs—for instance, an economic forecasting system that estimates changes in the Gross National Product, inflation, unemployment, etc. Assuming one has available a set of test cases where ground truth is known, then a simple method for "getting a feel" for the accuracy of the system's predictions is simply to plot a set of predicted and observed scores and to compare this plot with the ideal prediction line. For example, Figure 5-5 plots the data shown in Table 5-4. When compared to the ideal line, one can see that the predictive accuracy of the expert system is very high. Furthermore, by comparison to the ideal line, one can see that the system tends to overestimate the smaller values.

Table 5-4: Sample Data for Expert System Predicting Quantitative Values

Expert System Predictions	Observed Values
120	99
123	127
145	138
165	176
95	112
49	58
96	101
123	110
95	89
86	94
153	167
155	168
78	101
197	188
101	99

A more formal analysis of predictive vs. observed data can be achieved using a statistical procedure known as linear regression. Linear regression provides two useful outputs. The first output is the correlation between the predicted and observed values. The square of this value (often labeled R^2) is an estimate of the variance in the dependent variable (here the actual score) accounted for by the independent variable (here the predicted score). In the case of the data in Table 5-4, $R^2 = .8994$. The expert system's predictions seem to account for most of the observed variance. The second output is the regression line itself. This line is a "best fit" summary of the plot. If the regression line deviates from the ideal line, then this suggests some systematic deviations from the best prediction—not just random error. For instance, the regression line in Figure 5-5 has a positive intercept, suggesting again that the expert system is either overestimating low values or underestimating high ones.

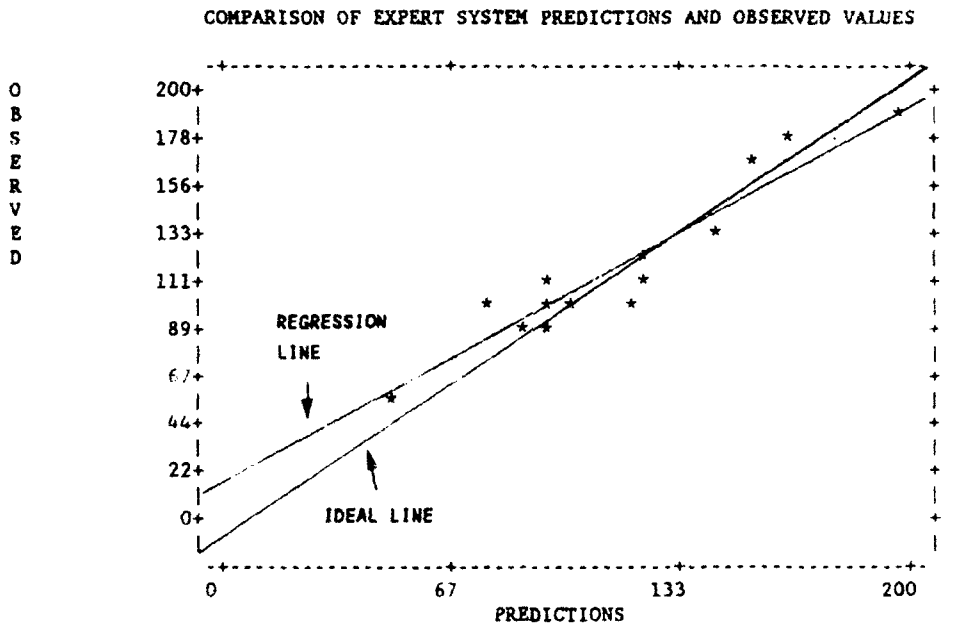


Figure 5-5: Plot of Predictions, Outcomes, and Regression Line for Table 5-4

Finally, if knowledge engineering is still ongoing, one might consider using the regression line as a basis for systematically modifying the knowledge base. Procedurally this can be achieved by adding to the pre-

dicted score the difference between the predicted score and the regression line scores. For example, if the score predicted by the expert system is 120, and the corresponding regression line score is 125, then the knowledge base should be adjusted so that it reports 125 where it used to report 120. If the linear regression analysis is based on a lot of data, then this modification to the knowledge base should increase performance accuracy.

CASE 4: COMPARISON TO EXPERT JUDGMENT

In the previous sections we made two important assumptions: first, that the performance objective of the expert system was to minimize error; second, that test cases could be generated where ground truth is not available. For many expert systems, however, the performance objective is not only to minimize error, but also to "capture" human expert judgment; that is, the system is designed to codify expert knowledge. When this is the case, an important evaluation criterion is the extent to which the expert system agrees with human expert judgment.

The problem of measuring expert judgment/expert system agreement is similar to Case 3 where one is assessing the accuracy of quantitative predictions. Here both the expert and expert system will express quantitative judgments (usually belief values) for each of a set of final and intermediate hypotheses. For instance, if the expert system evaluates whether hypothesis H or its negation is true, then one can compare the expert system's bel(H) values with those of one or more experts.

To illustrate this process, consider Table 5-5. Here the belief values of an expert system and three experts are summarized for twenty test cases. As before, one can perform a regression analysis comparing the system's judgments with each of the experts, or some averaging of the expert's opinions.

Table 5-5: Twenty Judgments from Three Experts and Expert Systems

Expert 1	Expert 2	Expert 3	Average of Experts	Expert System
25	63	52	46	76
86	82	53	73	77
94	113	93	100	98
163	140	162	155	168
166	129	110	135	123
50	55	69	58	66
52	63	69	61	77
58	12	53	41	84
94	99	84	92	95
163	187	140	163	122
112	45	96	84	90
198	129	160	162	144
44	32	53	43	67
13	36	62	37	48
125	92	100	105	100
146	163	120	143	127
72	92	121	95	89
132	69	123	108	121
163	94	126	127	129
153	152	155	153	113

Figure 5-6, for instance, compares the system's belief values with the average of the three experts' judgments. As we can see, the regression line deviates significantly from the ideal line, and there is a lot of fluctuation around this line. This indicates that there are significant differences between the experts' and expert system's judgments.

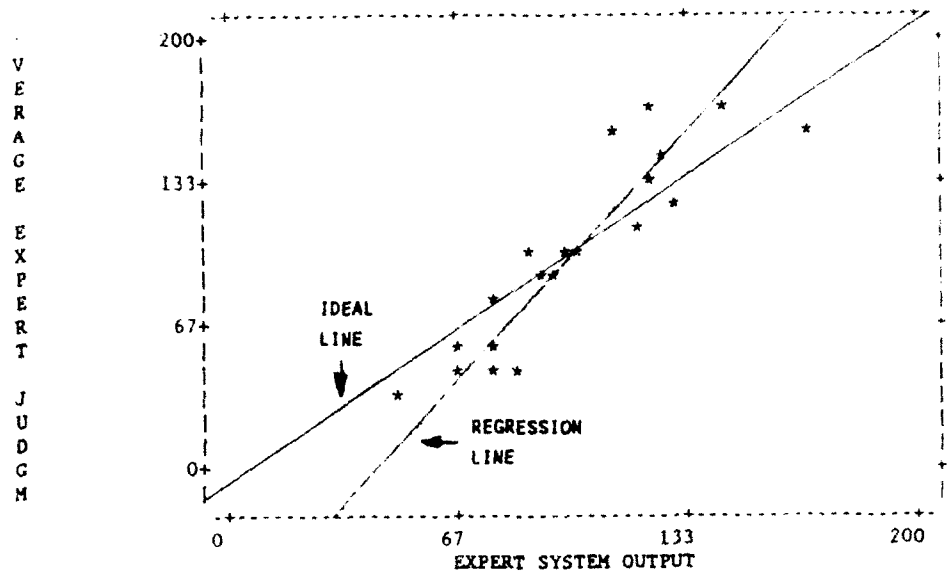


Figure 5-6: Comparison of Expert System and Average Expert Judgment

A more detailed analysis can be performed to examine the intercorrelation of the expert system's belief values and the various expert judgments. For instance, the set of intercorrelations for the data in Table 5-5 is found in Table 5-6. As can be seen, the correlation between the experts is about the same as the correlation between the expert system and the experts. This suggests that, although there are some differences between the expert and expert system judgments, these differences do not suggest that the expert system's judgments are outside the norm of expert opinion. In short, the overall evaluation of this system is that it reflects expert judgment. (Note, however, that it is still possible that a more detailed analysis will reveal some consistent differences.)

Table 5-6: Intercorrelation of Columns in Table 5-5

	Expert 1	Expert 2	Expert 3	Average of Experts	Expert System
Expert 1	1.0	.77	.87	.95	.91
Expert 2		1.0	.78	.91	.73
Expert 3			1.0	.94	.88
Average of Experts				1.0	.90
Expert System					1.0

SOME OTHER APPROACHES

In this chapter some procedures for evaluating the performance of an expert system were recommended. Before closing this chapter, we would like to mention some other approaches that are found in the literature, and explain why we do not recommend them.

Probability Scoring Rules. In the judgment and forecasting literature (as well as in the Bayesian network literature), probability scoring rules are commonly used to assess the accuracy of probabilistic judgments. The most commonly used scoring rule, initially proposed by Brier (1950) is the mean probability score (MPS) which is simply the average squared error of predictions vs. outcomes. For example, suppose that on three consecutive days a weather forecaster predicts a 20%, 60%, and 80% chance of rain. In fact, it rained only on the third day. Then

$$MPS = [(.2-0)^2 + (.6-0)^2 + (.8-1)^2]/3 = .147.$$

Our approach differs from the use of probability scoring rules in two ways. First, we have focused on measures that have a "behavioral" interpretation. P_U and P_E tell us something about how a user can use an expert system. In contrast, the behavioral implications of "MPS = .147" are unclear. Second, probability scoring rules measure the deviation of outcomes from the absolute belief values. This presupposes that the belief values are probability estimates. Furthermore, it may fail to measure discrimination. Note, for instance, that in Table 5-1, bel(ghypl) is almost always less than .5, even when ghypl is true. (Consequently, the MPS score for this expert system would be very low, even though the expert system effectively discriminates when ghypl is true vs. false.)

Comparison to Linear Models. Another approach (Levi, 1989) involves the use of a linear regression analysis to identify any linear relationships between the cues (evidence items) and (a) human expert judgments and (b) the correct diagnosis (see Levi, 1989, for discussion). The argument here is that an expert system is useful only if both (a) and (b) reveal a large nonlinear component, and that expert judgments effectively predict

the nonlinearity between the cues and correct diagnosis. If these conditions are not met, then a complex expert system could be replaced by a much simpler linear model. There is no "added value" in building an expert system. Although this added value approach clearly has merit, it really addresses an orthogonal issue. Although it may be possible to conclude that the expert system has "no added" value vis. a vis. a linear model, this does not impact performance. Performance remains the same even if we end up concluding that the extra cost incurred to build the expert system was not well spent.

Paired t-Tests. An alternative approach to compare human and expert system judgments (O'Keefe et al., 1987) is to use a paired t-test analysis. Here one does a node-by-node comparison t-test comparison to determine if there is a statistically significant difference between the human expert and expert system judgments. An even more stringent procedure is to simultaneously compare all the nodes using Hotellings' T^2 analysis. Although this approach has some merit, it should be noted that the statistical significance results, themselves, are not very useful. This is because, almost certainly, there *is some* difference between expert system and human expert judgment. And given that some difference does exist, one is almost certain to get a result that indicates a statistically reliable difference between the expert system and the human expert.

CHAPTER 6:

MORE ABOUT EMPIRICAL TEST AND EVALUATION METHODS

In contrast to (a) technical test and evaluation procedures, which focus on how well the expert system's knowledge base, inference engine, and service requirements were developed, or (b) subjective test and evaluation procedures, which focus on how much users liked the expert system, empirical test and evaluation procedures focus on how well decision makers performed the task with (versus without) the expert system. Remember, the expert system may be addressing only part of a larger organizational decision. And in many cases, the operator, who may be more or less accurate than the expert system's knowledge base, is using the system to support his/her decision making; consequently, the operator can override the expert system's recommendation. Therefore, even if the technical evaluation of the knowledge base shows that it has high predictive accuracy, the persons working with the expert system, and the larger organization of which they are a part, may or may not perform better with the expert system.

This chapter will explicitly consider experiments, quasi-experiments, and case studies. In particular, experiments are most applicable during prototyping and at later steps in the process after the expert system has been developed, packaged, and transferred to an (intermediate) test organization or the actual target organization. However, the randomization and sample size requirements of experiments are not always possible in the latter kinds of environments. Consequently, the second part of the chapter discusses the use of quasi-experimental and case study designs for objectively assessing the performance impact of expert systems in more operational versus developmental environments. As Yin (1984) has pointed out, quasi-experiments are appropriate in those settings where the logic of experimental design can be applied even though there is less experimenter control. In contrast, case studies are appropriate in settings where even the minimal experimental control required for quasi-experiments is not possible. In case study designs, as in all forms of empirical research, it is the logic of the research design linking the data collected—and the conclusions drawn from it—to the initial questions driving the study that determines the validity of the research. In our case, the goal

is to rule out rival hypotheses for explaining performance differences with (versus without) the expert system.

The purpose of this chapter is to help one perform better empirical evaluations. The discussion will be at a general level, for it is not presumed that this one chapter can even begin to cover all the details found in experimental design and statistical analysis texts. Rather, the goal is to sensitize the reader to the different issues that must be considered when attempting to empirically demonstrate the performance benefits of expert systems. Such a demonstration typically is (and in our opinion should be) essential for sponsors to approve the transfer of an expert system into an operational environment. After all, the bottom line is typically whether people can perform their tasks better, faster, and/or more cheaply with (than without) the expert system. That's what the sponsor wants to know. Yet, as Sharda et al. (1988) point out, the majority of claims regarding the benefits of decision support and expert system technology are based on studies that have not effectively controlled for alternative, plausible hypotheses to explain improved performance.

Again, from the perspective of the SHOR paradigm, the test and evaluation team's job is to help members of the sponsoring team decide whether the expert system is an effective option for dealing with hypotheses regarding the current and/or future problem environment with which the organization will be dealing. Remember, the initial decision to build the expert system was nothing more than a hypothesis that the expert system would improve the organization's decision making and, in turn, its performance. The test and evaluation team can use empirical evaluation methods to not only help members of the sponsoring team assess the adequacy of this hypothesis, but also identify what corrective actions to take if the expert system is found lacking. This latter point is extremely important from a prototyping perspective, for empirical feedback provides important guidance for improving the system. Correspondingly, the failure to identify whether or not an expert system actually caused improved performance, as suggested by the reference to Sharda et al.'s findings above, eliminates the opportunity to improve a deficient system.

The remainder of this chapter is divided into two parts. The first part identifies issues to consider when conducting experiments; the second part addresses quasi-experiments and case studies. In all cases, we are focusing on ground truth performance criteria, which are shaded in Figure 6-1. Our principal concern is on assessing the adequacy of an expert system on the three ground truth performance criteria: speed, accuracy, and bias. As you will remember, accuracy and bias also were the principal evaluation criteria when assessing the predictive accuracy of the knowledge base. Now, however, we are considering the performance of the person (or organization)/system unit.

It is important to note that we can also use subjective performance criteria as dependent measures in an experiment. That is, we could assess how well or fast experts thought they solved test problems with (versus without) the expert system. Such assessments are important for two reasons. First, ground truth performance measures are often difficult to obtain (or simply not obtained) in operational environments when a test is not being conducted. In such situations, users' judgments as to the speed and quality of the solutions generated with (versus without) an expert system often determine whether or not they will use the system. Second, empirical research (e.g., see Cats-Baril and Huber, 1987) indicates that ground truth and judgment measures of performance do not always agree. The test permits one to assess the correlation between ground truth and judgment measures of performance.

EXPERIMENTS

Experiments are, by far, the most common and commonly thought of empirical evaluation method. They are particularly appropriate when a number of people would actually use the expert system, for experiments are designed to help generalize from a test sample to the larger population, which, in our case, would be system users and their organizations. However, experiments also can be conducted even if the expert system is being developed for use by one person. In this case, for example, the prospective user could solve a number of representative problems with and without the expert system in order to assess whether the user performed better with the system.

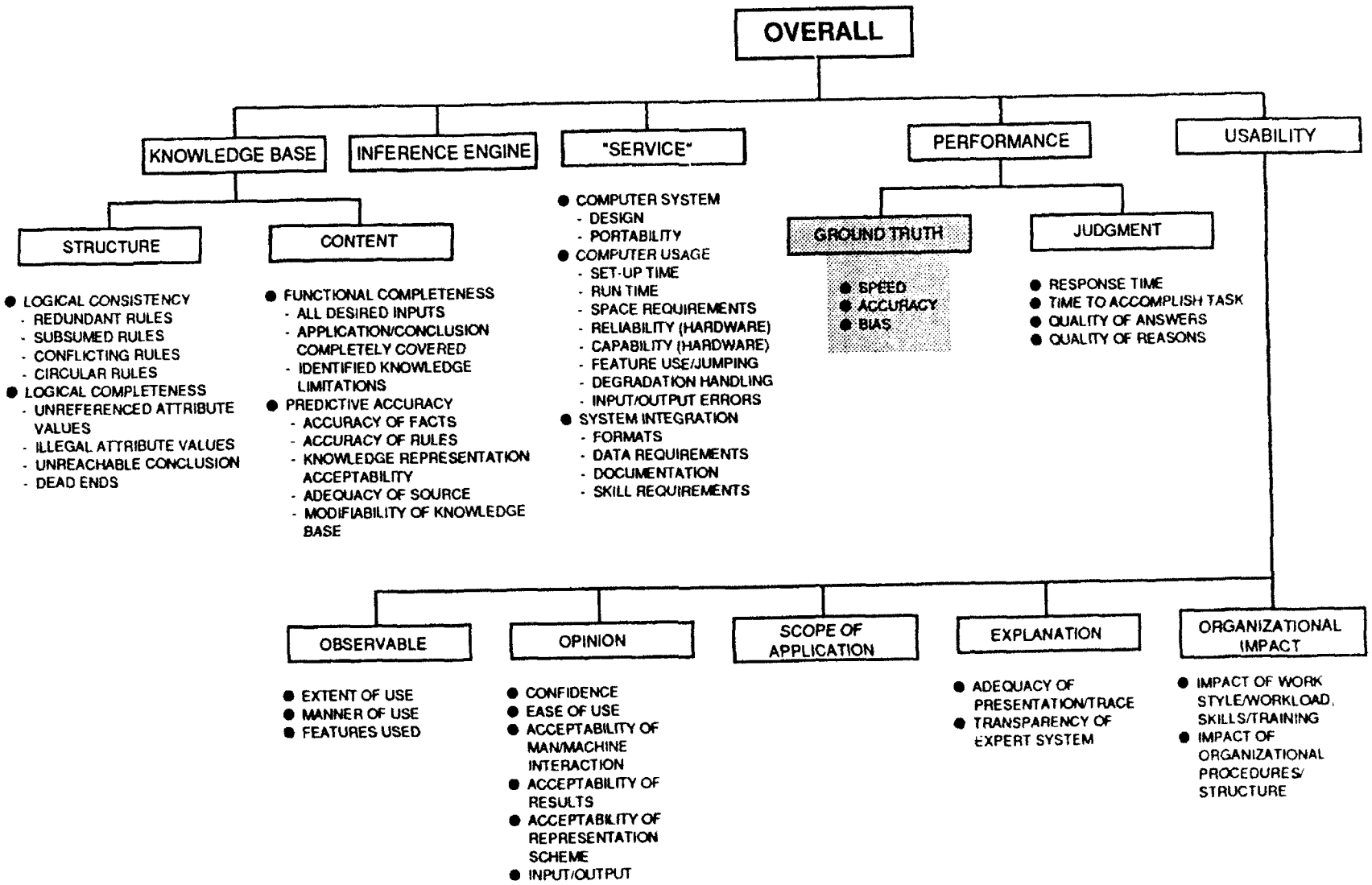


Figure 6-1: Evaluation Criteria being Assessed by Empirical Test and Evaluation Method

One typically thinks of two kinds of experiments. The first kind tests the expert system against objective benchmarks that often form performance constraints. When the expert system passes the benchmark, it proceeds further; when it fails, it undergoes further development or is set aside. "For example, it is not enough to know that with the aid the user can arrive at a decision in 30 min. If the organizational user required a decision in 30 min, the aid would be effective. If a decision was needed in 15 min, the aid would not be effective" (Riedel and Pitz, 1986, pp. 984-985).

First, it should be noted that such performance benchmarks differ from the more traditional time and efficiency measures used to benchmark computer systems. Time and efficiency benchmarks typically get developed during requirements analyses emphasizing a features-based approach. Although such performance constraints may be necessary in real-time, life-critical activities, they are unnecessary for many expert system applications. Consequently, we are not going to consider the timing and efficiency approach further. Readers interested in it are referred to Press (1989), who benchmarked different expert systems on the time required to load and execute different types of knowledge bases, and the amount of disk space required in source and fast-load formats. Although Press obtained empirical data assessing different systems, his focus was on the "service requirements" in our evaluation hierarchy, not our performance criteria.

Second, performance benchmarks, like those illustrated in the quote by Riedel and Pitz, represent noncompensatory decision rules; that is, performance on other evaluation criteria do not compensate for failing the performance benchmark. Such a position may, for example, be inconsistent with the compensatory decision rule guiding the sponsoring team's intuitive decision-making processes or more formal subjective methods, such as multiattribute utility technology. After all, it's quite possible that the sponsoring team might be willing to give up some time for task accomplishment (or some whatever) in order to gain even a little improvement on other MOEs, such as decision performance or personnel staffing requirements (or whatever).

The second kind of experiment, and the one that is focused on here, is a factorial design where (a) one or more factors are systematically varied as

the independent variable(s), and (b) the dependent variable(s) are quantitative, objective measures of system performance. As summarized in Table 6-1, there are five basic components of most factorial experiments. First, there are the participants in the experiment. (They are often called subjects or test subjects even though they are not the subject of the test.)

Table 6-1: Some Summary Comments About Experiments

Two Typical Kinds:

Benchmark Testing
Factorial Designs

Components of Factorial Experiments:

Participants
Independent Variables
Task
Dependent Variables (and measures)
Experimental Procedures
Statistical Analyses

General Approaches for Controlling Rival Hypotheses:

Include in Experimental Design
Eliminate Their Ability to Affect the Results

Second, there is the experimental condition(s) or independent variable(s) of interest, such as whether the participants perform the task with (versus without) the expert system. For example, there were two independent variables in the DART experiment: the "degree of support" and "problem type." There were two levels on the "degree of support" independent variable. The "unaided condition" was a control group; it represented the degree of support the decision maker had without DART. The second level was the "aided" condition; it represented the support level provided by DART. The "degree of support" variable need not be limited to two levels. Indeed, it might be extremely advantageous to evaluate two or more variants of the expert system, particularly during prototyping when there is substantial uncertainty about how sophisticated a system users need. And, in the DART experiment, there were two different problem scenarios because of the development team's concern

about DART's performance being sensitive to the characteristics of the scenario. And, as it turned out, it was.

Third, there is the task that the participants are to perform during the course of the experiment. The level of task difficulty should be either as representative of the operational environment as possible or matched to the hypothesized performance capabilities of the expert system. The capabilities of the expert system depend on its stage of development (e.g., see Gaschnig et al., 1983; Marcot, 1987).

Fourth, there is the dependent variable(s) (or MOEs) of interest. As we noted in the introduction to this chapter, we are interested in testing the expert system on one or more of the "ground truth" or "judgment" performance criteria in our evaluation hierarchy. Ground truth measures of accuracy and bias or, more globally, decision quality, depend on having correct answers. Judgments of decision quality can be made by either the participants in the experiment, who may or may not be experts in the substantive field depending on the proposed users of the system, or other experts. The judgments of experts is obviously the preferred judgment measure.

Ground truth measures are the preferred quality measures. If experts are used, attempts must be made to keep them blind as to which experimental conditions produced the solutions so that this information does not inadvertently bias their ratings. Such a context is often referred to as a Turing test. This was done in the DART experiment; see Chandrasekaran (1983) and Yu et al. (1979) for other examples.

And, fifth, there are the procedures governing the overall implementation of the experiment. Substantial care should be directed toward accurately representing the unaided as well as aided condition to ensure a fair test. If performance is better in the "aided" condition, we want to be able to say that it is due to the expert system and not some other extraneous factor. In order to do so, we need to (ideally) try to control for all "plausible rival hypotheses" (Campbell and Stanley, 1966, p. 36) that might explain the obtained findings. Toward that goal we introduce the concepts of reliability and validity. [Note: Statistical analyses—the sixth component

of experimental designs—will be considered later in this section of Chapter 6.]

Reliability and Validity Broadly Defined

Yin (1984, p. 36) defines reliability as "demonstrating that the operations of a study—such as the data collection procedures—can be repeated, with the same results. The key concept is replication. In contrast, "valid" is defined by Webster's dictionary (1966, p. 1608) as that which is sound because it is "well grounded on principles or evidence." If an experiment is valid, its conclusions can be accepted; that is, rival hypotheses have been controlled for.

An experiment can be reliable, but its conclusions invalid for numerous reasons that will be considered below. However, an experiment can not result in valid conclusions if it is unreliable; that is, one can not conclude that the results are well grounded if the evidence upon which they are based is undependable. Consequently, the basis for good experimentation is reliable (i.e., dependable) procedures and measures. Although far from trivial, reliability is typically possible in experimentation because of high experimenter control. The experimenter can pilot-test and subsequently modify the procedures and measures over and over again until they produce the same results when applied to the same situation, regardless of who performs the experiment. In contrast, there exist a number of threats to the validity of our conclusions regardless of the experiment's reliability.

There are four types of validity that need to be considered when performing experiments: construct, internal, statistical conclusion, and external. Each type is now discussed within the context of experiments, and again in the sections addressing quasi-experimental and case-study designs.

Internal Validity

Yin (1984, p. 36) has defined internal validity as "establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships." That is, we want

to be able to say that our independent variables, and not some other uncontrolled-for factors, caused the observed effects on our dependent variables. The "uncontrolled-for factors" represent rival hypotheses for explaining the experiment's results. Moreover, as Cook and Campbell (1979, p. 38) point out, "Internal validity has nothing to do with the abstract labeling of a presumed cause or effect; rather, it deals with the *relationship* between the research operations *irrespective of what they theoretically represent*" (italics theirs). This latter issue is considered under construct validity below.

There are two general approaches to controlling rival hypotheses, particularly those that might be considered spurious threats to causal inference. The first approach is to somehow include them in the design. One way to do so is to include rival hypotheses as other independent variables in the factorial design. For example, two different problem scenarios were used in the DART experiment in an effort to minimize the degree to which the results might be scenario-dependent. Another way to include rival hypotheses in the design is through the nature of the design itself. For example, Sharda et al. (1988) used a longitudinal, factorial design in order to examine whether time was a plausible hypothesis to explain (initial) performance decrements with decision support technology, a position consistent with a learning theory perspective. The point is that the one way to control for rival hypotheses is to use an experimental design that will explicitly permit one to test their effect on performance.

Although the first approach for controlling rival hypotheses is powerful, it has two limitations. First, it suffers from sample size limitations. The number of independent variables, the levels on these variables and, more generally, the experimental design itself need to be considered with care because of their implications for the number of participants required for statistical testing purposes. The larger the number of cells in the matrix representing the factorial design's pairing of levels on the independent variables, the more participants required to fill the cells and, thus, perform the experiment.

Second, it presumes that one knows all plausible rival hypotheses. This, however, is not possible; there may always be alternative hypotheses for

explaining the data. That is why the philosophy of science focuses on disconfirmation—not confirmation—of hypotheses. To quote Campbell and Stanley (1966, p. 34), "In a very fundamental sense, experimental results never 'confirm' or 'prove' a theory—rather, the successful theory is tested and escapes being disconfirmed ... An adequate hypothesis is one that has repeatedly survived such probing—but it may always be displaced by a new probe."

The second general approach for controlling for plausible rival hypotheses is attempting to eliminate any possibility that they can affect the results. In particular, one wants to control for extraneous factors that significantly impair our ability to make valid causal inferences. This second approach is exemplified by the concept of randomization. Webster's dictionary (1966, p. 1204) states that "random applies to that which occurs or is done without careful choice, aim, plan, etc.)." Arbitrarily assigning test subjects to the "aided" and "unaided" conditions in the above factorial design, or arbitrarily determining the order in which DART test subjects worked the test problems with or without the expert system, illustrates randomization.

It can not be overstated how important randomization is in experimentation. This point can be illustrated by considering the "threats to internal validity" that randomization eliminates as plausible rival hypotheses. To quote Cook and Campbell (1979, p. 56), "When respondents are randomly assigned to treatment groups, each group is similarly constituted on the average (no selection, maturation, or selection-maturation problems). Each experiences the same testing conditions and research instruments (no testing or instrumentation problems). No deliberate selection is made of high and low scores on any tests except under conditions where respondents are first matched according to, say, pretest scores and are then randomly assigned to treatment conditions (no statistical regression problems). Each group experiences the same global pattern of history (no history problem). And if there are treatment-related differences in who drops out of the experiment, this is interpretable as a consequence of the treatment. Thus, randomization takes care of many threats to internal validity." (Table 6-2 more formally defines the threats to internal validity considered above.)

Table 6-2: Definitions of (Selected) Threats to Internal Validity

Selection:	A threat due to the kinds of participants in one group versus another. [Note: Selection can interact with many of the other threats listed below.]
Maturation:	A threat due to participants gaining experience or in some manner changing during the course of the research.
Testing:	A threat potentially resulting because of the number of times participants' responses have been measured during the research.
Instrumentation:	A threat due to changing the way the dependent variables are measured during the research.
Statistical Regression:	A threat due to selecting participants for different (and particularly extreme) groups on the basis of pretest measures with less than perfect reliability.
History:	A threat due to an external event taking place during the course of the research that is not the treatment of interest.

True experiments include randomization. When randomization is not possible, one can employ *quasi-experimental designs*. However, quasi-experimental designs are not as effective as experimental designs in controlling extraneous factors. To quote Cook and Campbell (1979, p. 56), "With quasi-experimental groups, the situation is quite different. Instead of relying on randomization to rule out most internal validity threats, the investigator has to make all the threats explicit and then rule them out one by one. His task is, therefore, more laborious. It is also less enviable since his final causal inference will not be as strong as if he had conducted a randomized experiment. The principal reason for choosing to conduct randomized experiments over other types of research design is that they make causal inference easier."

Randomization is the best means for essentially equating the "aided" and "unaided" conditions prior to beginning the experiment. By doing so, it significantly limits the number of rival hypotheses that can be used to explain the obtained data, which in our case would (hopefully) be significantly better performance with the expert system. Randomization does not, however, rule out all threats to internal validity. In particular, Cook and

Campbell (1979, pp. 54-55) have identified four threats to internal validity that are not controlled for by randomization.

The first threat is the "diffusion or imitation of treatments" that may arise if members of the experimental and control groups can talk to each other during the course of the experiment and in some way obviate the potential effect of the treatment (e.g., the expert system) because of the information they communicate. The second threat is the "compensatory equalization of treatments" that may occur if administrators are reluctant to tolerate the perceived inequality between the experimental and control groups and, consequently, do not enforce the procedures distinguishing the two. The third threat is the "compensatory rivalry by respondents receiving less desirable treatments;" that is, members of the control group act to reduce or reverse the expected difference. Cook and Campbell note that this threat is particularly likely when intact units are assigned to different conditions or if members of the control group perceive themselves to be disadvantaged if the treatment condition is successful, both of which may happen in field tests of expert systems. The fourth threat is the "resentful demoralization of respondents receiving less desirable treatments," and represents the converse of the third threat.

Another threat not controlled by randomization is the unintentional confounding of the experimental treatment (e.g., the expert system) with some other factors. With this point in mind, we now turn to consider construct validity.

Construct Validity

Yin (1984, p. 36) has defined construct validity as "... establishing good operational measures for the concepts being studied." Good construct validity means that we are measuring that, and only that, which we want to be measuring. Of particular concern in empirical tests of expert systems is that the "with expert system" condition is not confounded by something else. If confounding exists, then the "something else" represents a rival hypothesis that could explain our obtained results.

The practice of giving control subjects placebos in medical research is a good example of trying to control for the possible confounding between the helpful concern of the physician and the chemical effects of the medication. Similarly, if one considers an expert system as analogous to a new medication, is the positive effect of a expert system the result of the DSS or the helpful concern of senior-level management? If it's the latter, performance will deteriorate once the expert system has been declared a success and its users are no longer the attention of upper-management scrutiny.

Possible confounding is also important when the system has a negative impact. For example, Markus (1984) used a case study design to show that the negative resistance to an implemented information system was not a function of the system's technical quality, but its mismatch with the organization's interaction patterns. Kaplan and Duchon (1988) used a case study and survey research approach to demonstrate that the response to an organization-wide information system was a function of users' perception of their jobs, not the system.

If we have some idea of what "other" variables may be confounded with our experimental conditions, then we want to either (1) take steps to eliminate their potential influence on our evaluation, (2) systematically incorporate them into our experimental design so that we can directly assess their effect, or (3) measure them so that we can perform a "post hoc" assessment. In choosing either of the latter two options, our goal is to measure each of these rival hypotheses (or constructs) in order to test their predictions with the collected data and, thereby, assess which hypotheses have been falsified. [Again, the perspective is on disconfirmation, not on confirmation, although the latter is how researchers typically report the implications of their findings.] More generally, the goal is valid causal inference. Since this is the goal of all empirical test and evaluation methods, not just experiments, the above points will be considered again when quasi-experimental and case study designs are addressed later in the chapter.

Statistical Conclusion Validity

In contrast to internal and construct validity, "[s]tatistical conclusion validity is concerned not with sources of systematic bias but with sources of random error and with the appropriate use of statistics and statistical tests" (Cook and Campbell, 1979, p. 80). The concern is with (1) whether the study is sensitive enough to permit reasonable statements regarding the covariation between independent and dependent variables, and (2) what constitutes appropriate tests of these statements.

There are two types of potential errors when performing statistical tests. The first type, called a Type I error (alpha), is the probability of *incorrectly rejecting the null hypothesis*, which is that there is no difference in the effect of the experimental conditions on the dependent variables. For an experiment assessing an expert system's effect on performance, it is the probability of incorrectly concluding that there is a difference in the performance levels obtained with (versus without) the expert system when, in fact, there is no difference. The second type of error, the Type II error (beta), is the probability of *incorrectly accepting the null hypothesis*. In our case, it is the probability of concluding that there is no difference in the performance levels obtained with (versus without) the expert system when, in fact, a difference exists. A test's statistical power is the complement of its Type II error level; that is, 1-beta. Statistical power is the probability that a statistical test will *correctly reject the null hypothesis*.

One wants to set up an experiment that appropriately balances the two types of errors. Such a balance is required because the beta and statistical power values are constrained by the value set for alpha. As Baroudi and Orlikowski (1988, pp. 88-89) point out, "The traditional belief is that the consequences of false positives are more serious than those of false negatives (Cohen, 1965). Therefore, Type I errors are usually guarded against more stringently. The distribution of risk between Type I and Type II errors, however, needs to be appropriate to the situation at hand. Mazen et al. (1987) present a graphic illustration of a case [the ill-fated Challenger space shuttle] where the risk of incurring a Type II error [saying there was

no problem when there was] far outweighed that for Type I [saying there was a problem when there wasn't]."

"Researchers who wish to conform to the convention of protecting themselves more against false positive claims should set alpha to .05 and beta to .20 (four times as much) (Cohen, 1977). Accepting these recommended values for alpha and beta results in a .80 value for power (1-beta), meaning that a statistical test having a power value of 0.80 has an 80% probability of detecting an effect if it exists. Cohen's prescription of a .80 conventional power level has become widely accepted as the norm ..." Cohen (1977), as well as other texts (e.g., Bailey, 1971) provide statistical tables for calculating statistical pow[illegible] on the basis of the alpha level, sample size, and predicted effect of the treatment.

Baroudi and Orlikowski (1988), Cohen (1977), Cook and Campbell (1979) and other researchers (e.g., see also Bailey, 1971) have discussed ways to increase the statistical power and, more generally, the "statistical conclusion validity" of one's experiments. Five will be considered here. Perhaps the most obvious way is to increase the sample size; that is, the number of people (or organizational units if that is the appropriate unit of analysis) participating in the experiment. The larger the sample size, the more precise the sample estimate of the values on the dependent variable for each condition, consistent with the Law of Large Numbers.

Second, attempt to increase the reliability of the experiment. For example, increase the reliability of the measurement instruments. As Cook and Campbell (1979, p. 43) point out, "... unreliability inflates standard errors of estimates and these standard errors play a crucial role in inferring differences between statistics, such as the means of different treatment groups." Similarly, increase the reliability of the procedures for implementing the different conditions in the experiment. By standardizing how people receive the treatments, one will decrease the error variance and, thereby, increase the probability of detecting true differences.

Third, give careful consideration to the research design. As Kraemer and Thiemann (1987) suggest, include only factors that are necessary to the

research question, or that have a documented and strong relation to the response. Including marginally relevant factors decreases statistical power if not appropriately compensated for by an increase in the sample size. In addition, try to allocate an equal number of test subjects to each condition. As Baroudi and Orlikowski (1988, p. 101) point out, "... if the group sizes are unequal, attenuation of observed effect sizes can occur, which potentially undermines the statistical power of the analysis, regardless of the total n." Also, if possible, use repeated measure designs, like the one in the DART evaluation. By repeatedly measuring the test subjects, one is able to partition out the error variance due to individual (versus treatment) differences and, thereby, increase the statistical power of the test.

Fourth, give careful consideration to the research question. As discussed above, consider which of the two types of error is most important in your experiment, and proceed according. Also, consider whether your hypothesis is directional or not. It might be, for example, that a sponsor will consider implementing an expert system only if it can be demonstrated that it improves performance during its operational test; poorer or even equivalent levels of performance to those achieved without the system are, for whatever reason, unimportant. In such a case, it is possible to increase the power of the test simply by moving from a two-tailed to one-tailed test, for a one-tailed test with an alpha of .05 has the same power as a two-tailed test with an alpha of .10, all other things being equal.

Fifth, consider the "effect size" that is of utility to the sponsors. "Effect size" is the difference in the performance levels achieved by the different conditions that is of scientific significance or value. The larger the "effect size" that is of importance (e.g., performance improvements of 100% vs. 10%, or 3 vs 1/3 standard deviations), the smaller the sample size required to find the effect. Lehner and Ulvila (1989) have shown that a surprisingly small number of test cases is required to test high utility performance enhancements for expert system users.

Thus far, we have not indicated what statistical tests should be performed; we have simply focused on the more general issues inherent in considering statistical conclusion validity. Moreover, we are hesitant to go into

much detail about statistical tests because we know that adequate consideration of them takes substantially more space than is available here. Nevertheless, consistent with the approach taken by O'Keefe et al. (1987), we briefly overview two classes of statistical tests. [Note: These tests are listed in Table 6-3 for summary purposes, as are the definitions of Type I and Type II error and the general approaches to increasing statistical power.]

Table 6-3: A Summary of Issues Involved in the Discussion of Statistical Conclusion Validity

Definitions:

Type I Error:	The probability of *incorrectly rejecting* the null hypothesis that there is no difference between test conditions.
Type II Error:	The probability of *incorrectly accepting* the null hypothesis that there is no difference between test conditions.
Statistical Power:	The probability of *correctly rejecting* the null hypothesis that there is no difference between test conditions.

General Approaches to Increasing Statistical Power:

Increase sample size
Increase the reliability of the experiment
Give careful consideration to the research design
Give careful consideration to the research question

Illustrative Types of Statistical Tests:

For one independent variable with two levels and one dependent variable:

Two-sample t-test

For more than two levels on one independent variable or two (or more) independent variables, but one dependent variable:

Analysis of Variance (ANOVA) with Planned or "post hoc" statistical tests

For one independent variable with two levels and two dependent variables:

Hotelling's one-sample T^2 test

For more than two levels on one independent variable or two (or more) independent variables, but two dependent variables:

Multivariate Analysis of Variance with ANOVAs and planned or "post hoc" statistical tests

For the first class of tests, assume that one has only one factor—the degree of aiding. Moreover, assume that one has only two levels on this factor, whether the participant worked the problem with or without the expert system, and one dependent variable (i.e., performance MOE). We can use a paired (or two-sample) t-test to assess whether the difference in the average performance levels obtained in the two conditions is significantly different. If we have more than two levels on this factor, or more than one factor, then we should use an Analysis of Variance (ANOVA) test instead of multiple t-tests in order to appropriately control for finding differences between our conditions by chance. The ANOVA should be accompanied by either "planned" or "post hoc" statistical tests of the (average) performance levels obtained in the different conditions depending on whether the observed differences were hypothesized or not prior to conducting the experiment.

There are, of course, many situations where there are multiple, not just one, dependent variables of interest. The second class of statistical tests deals with this case. Again, consider the case where we have only one factor, degree of support, and two levels on it. To quote O'Keefe et al. (1987, p. 87), "*While a paired t-test is appropriate when systems produce a single final result, simultaneously applying a paired t-test to a number of final results is inappropriate since we can expect the final results to be correlated ... In such cases, Hotelling's one-sample T^2 test should be used.*" In the case where there are more than two levels on the factor or multiple factors, then one should use a multivariate analysis of variance (MANOVA). If the MANOVA shows a significant difference between experimental conditions, then separate ANOVAS and planned and post hoc comparison tests can be performed to statistically examine the data.

External Validity

In addition to internal validity, construct validity, and statistical conclusion validity, one also needs to consider external validity. To quote Campbell and Stanley (1966, p. 5), "*External validity* asks the question of *generalizability*: To what populations, settings, treatment variables, and measurement variables can this effect be generalized?" [italics theirs]. Within the context of most expert system tests and evaluations, external

validity deals with the extent to which the results obtained in an experiment conducted in a simulated (laboratory) setting will generalize to operational environments.

As pointed out in Chapter 1, one of the most fundamental dimensions over which test settings can vary is their degree of fidelity to the target setting. The simulated environment, the simulated decision-making organization, and even the simulated user can range between being only superficially accurate to being accurate in great detail. By itself, high fidelity is desirable in any evaluation setting, but it is expensive. Besides increased dollar costs and evaluation time, fidelity introduces an additional cost in terms of loss of experimenter control as one moves from the laboratory to the operational environment. This means that it may be increasingly difficult to obtain the desired measures in the latter context. Moreover, even with well implemented quasi-experimental and case study designs, these measures will be increasingly susceptible to influences that are extraneous to the causal relations of concern, thereby representing threats to internal and construct validity. Consequently, a tradeoff is established between fidelity and costs such that, depending on the objective of the test and evaluation, it may be desirable to simulate all parts of the target setting prior to moving the assessment into the operational environment.

As mentioned earlier, experimentation during later software and hardware development steps is a natural part of the prototyping process. To date, this experimentation typically focuses on the Expert System/User interface (e.g., see Gould and Lewis, 1985) via "human factors evaluations" (Riedel and Pitz, 1986, p.990). While essential to developing a well-liked, usable system, such experiments typically have low-fidelity User-Expert System/Decision Making Organization (DMO) and DMO/Environment interfaces. Many expert systems are, however, used in organizations for the purposes of improving organizational decision making and, in turn, organizational performance. Consequently, increasing the fidelity of the organizational and environment interfaces is essential in generalizing the performance results obtained in the laboratory to a real-world setting.

In order to increase the fidelity of the User-Expert System/DMO and DMO/Environment interfaces, identify the organizational structures, processes, and communication patterns (both formal and informal) impinging on performing the task. This includes time pressures, interruptions, the reward structure, and even whether the decision maker or a subordinate will operate the expert system If a task is really performed by a group and not the organizationally identified, individual decision maker, then, ideally, the group needs to be represented in the laboratory. For some tasks, such as distributed ones, it may be possible to simulate the effects of a group by the information presented to the participating decision maker. For other group decision-making tasks, it may be possible to train members of the research staff to play certain roles. However, accurate representation of the groups for some tasks may require the presence of trained personnel performing their parts of the task in order to assess the value of the expert system. Such a situation will affect the physical size and structure of the laboratory setting. In addition, audio- and video-taping capabilities should be used as a data collection mechanism if the interactions among group members are hypothesized to play a significant role in the successful performance of the group.

In addition, try to make the interaction with the expert system as representative as possible of that which would occur in the actual environment. This includes the training in using the expert system. Insufficient training will result in the experiment being an unreliable measure of the potential value of the expert system; therefore, if an error is made, err on the side of too much versus too little training. If possible, use objective measures to demonstrate that the user has been trained to the desired level of proficiency on the system prior to beginning the experimental session. Also, make sure that training includes working representative problems. Not only will users be better trained to participate in the experiment, they will be better able to subjectively evaluate the expert system's strengths and weaknesses.

From both an experimentation and prototyping perspective, the fidelity of both the organization and environment interfaces should be improved systematically, as the user interface is typically done now, in order to provide maximum experimenter control for assessing the characteristics of each

that have the greatest impact on expert system performance. Such a perspective is, however, idealistic, given the time and cost constraints on most expert system development efforts. A more practical approach for later in the development cycle is to develop a gaming simulation that is as representative as possible of the User-Expert System/Organization and Organization/Environment interfaces. [Note: As a result of the requirements analysis, essential features of both interfaces should be kept in mind throughout the early steps of the development process.] In particular, a two-phased experimentation approach could be implemented.

The first phase could be a relatively straightforward experiment testing whether or not the expert system significantly improved objective process and performance MOEs. A positive finding in as representative a "simulated" setting as possible, particularly over variations in representative problem scenarios and personnel, would provide the empirical results necessary for attempting to validate the expert system in its target setting. A negative finding would lead to the second phase, which would be experimentation oriented to ascertaining (a) why performance was not significantly better with the expert system, if that could not be assessed during the Phase 1 experiment, and (b) whether modifications to the system result in improved performance. The second phase might also include modifications to aspects of the organization that are hypothesized to affect performance with the expert system. More generally, the goal of the second phase of experimental testing is to better assess the factors that are affecting expert system performance and attempt to rectify them in as representative an experimental setting as possible before transferring the system into its operational (or operational test) environment.

Field Experiments

Once the expert system has demonstrated superior performance in a representative, laboratory setting, it is ready for an experiment in the actual target setting. Although more difficult to implement, "field experiments" also need to control for all threats to validity. For example, regarding internal validity, organizational units (e.g., divisions/sections in a large company or governmental agency) would be *randomly* assigned to the

"with expert system" and "without expert system" conditions, and their performance measured after it has stabilized. The unit of analysis is the performance of the organizational unit; consequently, a large enough sample of units would be required for performing statistical tests.

Regarding construct validity, attention must be directed toward tightly measuring the process and performance variables of interest and controlling other variables that might be confounded with them. If we have some ideas of what these variables (or rival hypotheses) may be, we want to take steps to either eliminate their potential impact on the study, systematically incorporate them into our experimental design, or measure them so that we can perform a "post hoc" assessment. If possible, a "placebo" condition would be included too.

Regarding statistical conclusion validity, we want to have as high a level of statistical power as possible. This statement implies serious consideration of our research design, the questions we are addressing (including effect size), and the relative importance of Type I and Type II errors. The issue of external validity does not have to be addressed if, and only if, the field test includes all aspects of the population to which we want to generalize our results.

CASE STUDIES AND QUASI-EXPERIMENTS

The sample size and randomization requirements of true experiments are not possible in many organizations. Case studies and quasi-experiments should be used in such situations. To quote Campbell and Stanley (1966, p. 34), "There are many social settings in which the research person can introduce something like experimental design into his scheduling of data collection procedures (e.g., the *when* and *to whom* of measurement), even though he lacks the full control over the scheduling of experimental stimuli (the *when* and *to whom* of exposure and the ability to randomize exposures) which make a true experiment possible. Collectively, such situations can be regarded as quasi-experimental designs."

All four types of validity need to be considered for these designs just as for experiments. The reduced control of not being able to perform field experiments makes empirical evaluations in the actual setting more difficult, but it does not eliminate our ability to perform them consistent with the tenets of the scientific method. "[T]he core of the scientific method is not experimentation per se, but rather the strategy connoted by the phrase [evaluating] 'plausible rival hypotheses'" (Campbell, 1984, p. 7).

There are different types of quasi-experimental designs. Campbell and Stanley (1966) identify ten types, not counting variations of these themes. The three quasi-experimental designs considered below are: (a) time-series designs, where the organizational unit is measured for a period of time before and after the treatment (e.g., expert system implementation); (b) multiple time-series designs using a control group; and (c) nonequivalent control group designs that obtain pretest and posttest measures at only one time for a nonrandomized sample of treatment and control groups.

Quasi-experimental designs represent a substantial advance over the "pre-experimental designs" (Campbell and Stanley, 1966) found in many "field studies" (Sharda et al., 1988). We will first overview the three pre-experimental designs and then discuss how case study and quasi-experimental designs represent improvements over them.

Pre-Experimental Designs

Campbell and Stanley (1966, p. 6) called the first pre-experimental design "the one-shot case study." Cook and Campbell (1979, p. 96) later renamed this "the one-group posttest-only study" in order that it not be confused with appropriately conducted case study designs. In this approach, one organizational unit is given the treatment (e.g., the expert system) and performance is subsequently measured. There is no pretesting and there is no control group; instead, a "... single instance is implicitly compared with other events casually observed and remembered. The inferences are based upon general expectations of what the [performance] data would have been had the [treatment] not occurred, etc." (p. 6).

The one-group posttest-only study violates all four types of validity. First, there is no control for (or even consideration of) internal validity threats due to how participants are selected or what may have also occurred during the treatment stage, either through historical events or other changes to the participants or their organizational context. Second, nothing is measured so it is impossible to determine what extraneous factors may have been confounded with the treatment, or to assess their effects. Third, there is no explicit measurement of performance variables or comparison with another group; consequently it is impossible to assess statistical conclusion validity. And, fourth, it provides no justified basis for predicting the effect of the treatment on another group of participants.

The second pre-experimental design is the one-group pretest-posttest design, where implementation of the expert system represents the treatment. The problem with this design is that it does not control for the effect of other plausible hypotheses that could have improved performance between the pretest and the posttest. Three types of plausible rival hypotheses immediately come to mind. First, there may have been "other events" that occurred between the two tests that can explain the results—that is, history. Second, if the selected group represented extreme performers (e.g., the very best or very worst), then one would expect pretest-posttest differences to be affected by statistical regression to the mean. And, third, the design does not rule out other effects that are confounded with the treatment, such as the "special attention" that goes with the implementation of the expert system.

The third pre-experimental design is the static group design, also called the "posttest-only design with nonequivalent groups" (Cook and Campbell, 1979, p. 98), where the subsequent performance of the group receiving the treatment is compared with that of a group without the treatment. Since there is no pretest or randomization with this design, there is no "... formal means of certifying that the groups would have been equivalent had it not been for the treatment [e.g., the expert system]" (Campbell and Stanley, 1966, p. 12). Or, to be more blunt, "The plausibility of selection differences in research with nonequivalent groups usually renders the design uninterpretable" (Cook and Campbell, 1979, p. 98). The last two pre-experimental designs are somewhat insidious in the sense that, on the surface, they represent "pretest-

posttest control group" and "posttest-only control group" designs, respectively, which are true experimental designs because subjects are randomly assigned to at least two conditions—the treatment and the control groups.

Explicit case study and quasi-experimental designs will now be considered in relation to the three pre-experimental designs considered above. In particular, case study designs will be juxtaposed to the one-group posttest-only design; time-series designs will be juxtaposed to the one-group pretest-posttest design; and nonequivalent control group designs will be juxtaposed to the posttest-only design with nonequivalent control groups. The reader should keep in mind that we are sampling only from a wide variety of quasi-experimental designs. Although they are considered together here, case study and quasi-experimental designs represent different evaluation approaches.

Appropriate Case Studies

Yin (1984, p. 23) has defined a case study as "an empirical inquiry that investigates a contemporary phenomenon within its real-life context; when the boundaries between phenomenon and context are not clearly evident; and which multiple sources of evidence are used." This definition nicely fits the evaluation of an expert system or, more broadly, any form of information technology (or intervention) in an operational environment. As Lee (1989, p. 33) points out, "There is a strong case-study tradition in the academic field of management information systems ... [Our concern is] to clarify the methodological basis upon which to conduct case studies."

We first consider construct validity, which is the attempt to ensure that one is measuring the concept that one wants to measure. To quote Yin (1984, p. 37), "To meet the test of construct validity, an investigator must be sure to cover two steps: (1) select the specific types of changes that are to be studied (in relation to the original objectives of the study) and (2) demonstrate that the selected measures of these changes *do indeed reflect* the specific types of change that have been selected" [italics ours]. The reason for the italics is that the lack of experimenter control in case study research makes it much more difficult than in an experiment to minimize the

potential confounding of the treatment of interest (e.g., the expert system) with other variables (e.g., special attention).

Moreover, lack of control makes it more difficult to be sure one is measuring that which one wants to measure. For example, in a laboratory experiment it might be quite easy to measure decision quality because one has a ground-truth solution to the problem scenario. In contrast, in an operational setting, it might be quite difficult to define decision quality because it might be in the eye of the beholder. In such settings it is necessary to have multiple experts, preferably ones who are not part of the organization, rate decision quality and then use either a consensus position or an average rating to resolve any differences of opinion. In an effort to make the experts blind to the solution generated by the expert system, one should attempt to state the problem generally and to embed the actual solution in a range of hypothetical solutions.

The principal approach to solving the threat to construct validity in case study research is two-fold. First, the constructs need to be reliably measured. Again, the essence of reliability is replication—that is, the position that if another researcher did the same study a second time, one would obtain the same results. To quote Yin (1984, p. 40), "Note that the emphasis is on doing the *same* case over again, not on 'replicating' the results by doing *another* case study ... One prerequisite for allowing this other investigator to repeat an earlier case study is the need to document the procedures followed in the earlier case. Without such documentation, even you could not try to repeat your own work (which is another way of dealing with reliability) ... The general way of approaching the reliability problem is to make as many steps as possible as operational as possible, and to conduct research as if someone were always looking over your shoulder" [italics his].

Second, case study research should use multiple sources of evidence to measure the independent variable constructs. More generally, there should be a high correlation among different pieces of evidence all supposedly measuring the same construct. Conversely, there should be no correlation among pieces of evidence measuring different independent variable constructs. This is the concept of convergent and discriminant validation introduced by Campbell and

Fiske (1959). Quite simply, the idea is that measures of the same thing should be highly related; measures of different things should not be. For example, if the implementation of an expert system were preceded by effective training, one would expect that to be reflected in a number of ways, such as by (1) behavioral measures of proficiency in using the expert system prior to testing (or during implementation), and (2) subjective responses to interview questions. That is, different measures of "training proficiency" would all converge on the same result. Moreover, there should be no confounding with other independent variable constructs. For example, there should be no systematic relation between measures of training proficiency and organizational size or management attention if there is high construct validity.

It is important to emphasize that causal inference has not yet been considered; construct validity addresses only whether one is actually measuring the variables one wants to measure, not whether there are causal relations between these variables. In order to do the latter, we need to consider statistical conclusion validity and internal validity. The former is typically not possible in case study research because as Lee (1989, p. 35) points out, "... the study of a single case commonly yields more variables than data points—a situation that renders inapplicable the statistical controls of statistical experiments." Moreover, many case studies generate only qualitative, not quantitative data. Consequently, there is no way to perform a statistical test of the reasonableness of one's causal inferences.

Although one may not be able to perform statistical tests to determine statistical conclusion validity, one can still address the internal validity of case study research. At a more conceptual level, Lee (1989, p. 40) has pointed out that "... it must first be emphasized that mathematics is a subset of formal logic, not vice versa. Logical deductions do not require mathematics. *An MIS case study that performs its deductions with verbal propositions (i.e., qualitative analysis) therefore only deprives itself of the convenience of the rules of algebra; it does not deprive itself of the rules of formal logic, to which it may therefore still turn when carrying out the task of making controlled deductions*" [italics his].

Yin (1984, p. 105) has emphasized the use of three modes of data analysis for case study research: pattern matching, explanation building, and time-series analysis. [Note: "Time-series analysis" will be considered as a quasi-experimental design later in this chapter.] In its strongest form, pattern matching "... requires the development of rival theoretical propositions articulated in operational terms. The important characteristic of these rival explanations is that each involves a pattern of independent variables that is mutually exclusive: If one explanation is to be valid, the others cannot be. This means that the presence of certain independent variables (predicted by one explanation) precludes the presence of other independent variables (predicted by a rival explanation). The independent variables may involve several or many different types of characteristics or events, each assessed with different measures and instruments." Indeed, consistent with the above discussion regarding the use of multiple pieces of evidence for construct validation purposes, causal inference is enhanced if one uses different types of measures to support one hypothesis versus its plausible rivals. The MIS case study by Markus (1983) is an excellent example of the use of this mode of analysis.

In contrast to pattern matching, explanation building relies on iteration, where an initial set of propositions is compared with the obtained data, and subsequently revised and tested. The final explanation is seldom stipulated at the start of the study. Rather, it develops as the data are examined from a different perspective, one that emerges from the analysis itself. Obviously, such an approach can be dangerous if it relies on building a myopic chain of evidence focusing on support for this "new" hypothesis. Falsification, not confirmation, must be the goal; consequently, rival plausible hypotheses must be pitted against one another. The case studies by Bourgeois and Eisenhardt (1988) and Kaplan and Duchon (1988) are excellent examples of this second mode of analysis.

It should be noted that both modes of data analysis require that plausible rival hypotheses be *known* so that they can be evaluated against the data. This is, as we pointed out before, a difficult task. Moreover, it is a weakness compared to true experiments where randomization can be used to

control for the spurious effects of many (but not all) *unknown* hypotheses. Nevertheless, they are powerful modes of deduction.

Lastly, we consider the external validity of case study research. As was discussed earlier in this chapter, external validity refers to the generalizability of the obtained findings. Case studies have been routinely criticized on external validity grounds for, the argument goes, how can one generalize from a sample size of one? Yet, the same criticism can be leveled at a researcher attempting to generalize from a single experiment, for it would be just as precarious. Experiments and case studies are no different in terms of their external validity requirements. Both depend on theoretical propositions being tested and not falsified under various empirical conditions in order to assess how generalizable they are. The more conditions for which the predictions of a theory hold, whether these theoretical propositions are tested by experiments or case studies, the greater the external validity of the theory. The same holds for an expert system or any form of information technology. The broader the types of operational conditions for which the expert system *enhances* performance, the more confidence one has in its ability to enhance performance in subsequent settings.

Time-Series Designs

We now turn to consider time-series designs, the first of the two quasi-experimental designs considered in this chapter. As you will remember, time-series designs were juxtaposed to the one-group pretest-posttest design where, in our case, implementation of the expert system represents the treatment. The problem with this pre-experimental design is that it does not in any way control for the effect of other plausible hypotheses that could have improved performance between the pretest and the posttest.

The "simple interrupted time-series design" (Cook and Campbell, 1979, p. 209) uses the group itself as a partial control for alternative hypotheses by measuring the performance of the group *repeatedly* both before and after the treatment intervention. For example, if repeated measurements had shown that a group's performance was increasing linearly by 3 points on every observation, then it would be inappropriate to conclude that the expert system had a

positive affect on performance because the posttest was 3 points above the pretest. Similarly, if performance is known to vary with known cycles or actions, such as the time of year or the change in administrations, it would be inappropriate to assume that the expert system had significantly affected group performance without first accounting for these known causes of performance regularities.

The construct validity of a simple interrupted time-series design again depends on the extent to which one is measuring what one wants to measure. Like case studies, one should use multiple pieces of evidence to measure other variables that might be confounded with the expert system's implementation, as well as all dependent variables. Again, the focus should be on the convergent and discriminant validity of these measures; measures supposedly measuring the same construct should be correlated, while those measuring different constructs should not. Moreover, as Cook and Campbell (1979, p. 231) point out, "... data needs close scrutiny. Operational definitions need to be critically examined, and one should not assume that the construct label applied to a particular measure is necessarily a good fit. Inquiries have to be made about shifts in definition over the time the record is kept; where possible, the nature of the shift needs documenting." Also, special attention must be given to the reliability of the measures. Unreliability adds error into the measurement process, thereby reducing one's ability to find differences between the pretest and posttest observations.

The issues regarding Type I and Type II errors and statistical power that were discussed regarding the statistical conclusion validity of experiments are just as appropriate to quasi-experiments. For the simple interrupted time-series, this means that one wants to be able to make a reasonable statement as to whether the posttest observations represent a different pattern from the pretest ones. As the above "3-point example" illustrates, one should not use traditional statistical tests to assess mean differences. And, as McCain and McCleary (1979, p. 234) point out in their review paper of different statistical methods for performing time-series analyses, one should not use ordinary least squares (OLS) regression. "OLS regression requires an assumption that residuals, or error terms associated with each time-series observation, be independent. When naturally occurring

events or behavior are observed repeatedly over time, however, events closer to each other in time tend to be more correlated with each other than with events further removed in time. Since time is the independent variable of an OLS time-series regression, it follows that the error terms of consecutive observations are usually correlated ... [Consequently,] *the estimates of standard deviations (and hence, of significance tests) are biased*" [italics theirs].

There are numerous texts on time-series analysis and computer programs to help one perform it; consequently, we will not go over the different methods here. Indeed, consistent with our earlier discussion of statistical tests for experimental designs, we are hesitant to discuss time-series analysis methods in any detail because adequate discussion of the topic takes considerably more space than is available here. Instead, consistent with the detailed presentation by McCain and McCleary (1979), we will here only enumerate the four basic steps in the analysis for a simple interrupted time-series design.

The first step is called "identification." The goal is to identify the systematic component in the data that is not dependent on the treatment (i.e., the expert system). The systematic component is responsible for the correlation (called "autocorrelation") in the data independent of the treatment. When the autocorrelation structure is known, it can be explicitly incorporated into the model so that one can calculate unbiased estimates of the standard deviations and, thereby, statistically test the treatment's effect on the residuals.

The second step is "estimation;" that is, once a likely model has been identified, its parameters are estimated with programs using appropriate nonlinear equations. The third step is "diagnosis." The autocorrelation and partial autocorrelation of the residual terms are examined to diagnosis the adequacy of the estimation model. The goal is to conclude that the residuals are unbiased, essentially behaving as white noise. If they are, then one has an adequate model for predicting regularities in the data, independent of the treatment. If they don't, then one repeats the process of identification, estimation, and diagnosis until an acceptable model is found.

When one has a predictive model with unbiased residuals, one can proceed to "intervention hypothesis testing." In this step, one adds an intervention component to the model. The intervention component represents the hypothesized effect of the treatment to the model; it is represented by a "transfer function." For example, if the hypothesized effect is that of an abrupt, constant change, it would be represented by a step function. Gradual, constant change can be represented by a linear function. "If the intervention component increases the model's predictability, the parameters of the intervention component will be statistically significant ... Expressing the hypothesis testing component of time series in this way illustrates that the statistical analysis does not by itself test 'cause.' It asks only whether a statistically significant change in the series takes place at a specified point in the series. No explanations for the change are evaluated" (McCain and McCleary, 1979, p. 262).

When all is said and done, the simple interrupted time series is a weak design because of a number of threats to its internal validity, not statistical conclusion validity. The most obvious and significant threat to internal validity is that some simultaneous event other than the treatment caused a change in performance. Cook and Campbell (1979, p. 211) have referred to this as a "main effect of history." Another internal validity threat is "instrumentation." As they point out, administrative changes are sometimes accompanied by changes in record keeping. Since it is not uncommon for the implementation of an expert systems to cause administrative changes as well, changes in record keeping is a plausible rival hypothesis. And, third, "selection" could be an internal validity threat if the implementation of the expert system were also accompanied by a shift in the composition of the test group. [Note: It is assumed here that the threat to internal validity posed by seasonal or cyclical impacts on performance have been controlled for through the time-series analysis; otherwise, they pose a threat to internal validity.]

In an effort to control for these threats to internal validity, Campbell and Stanley (1966) advocated the addition of a control group to the time-series design. The control group should be comparable to the treatment group but, as Cook and Campbell (1979) demonstrated, attempting to match the groups at the point of the intervention can be difficult and sometimes cause spurious

effects. What is most important is that the control group be similar to the treatment group in the sense that it can be subjected to the same historical, instrumentation, and selection effects. Consequently, a significant shift in the posttest versus pretest observations of the treatment group, but not the control group, would disconfirm the above rival hypotheses compared to the hypothesized treatment effect.

The threats to external validity for the simple interrupted time-series design still exist when a control group is added to the design. Again, this is because external validity has to do with the generalizability of the results to settings and groups different from those in the test. The broader the range of settings for which the results hold, the better the time series' (or experiment's or case study's) external validity.

Nonequivalent Control Group Design

This quasi-experimental design was juxtaposed to the posttest-only design with nonequivalent groups. The latter is a pre-experimental design because the treatment and control groups are compared only on a posttest. The nonequivalent control group design adds a pretest measure for both groups in an effort to control for factors, other than the treatment, that might affect performance.

The nonequivalent control group design is similar to the time-series design with a control group, but it utilizes only one pretest and one posttest observation, a situation not uncommon in operational environments. As a result, however, it is not as effective at controlling for threats to internal validity as the "time-series with a control group." Moreover, it requires that a number of treatment and control groups be sampled because it uses only one pretest and posttest score per group. Although different types of measurements (i.e., multiple pieces of evidence) can be used to obtain this pretest and posttest score, one has a significantly different type of situation from that occurring in a time-series design where there are numerous, repeated pretest and posttest scores (using the same measurement instruments) over time. Consequently, nonequivalent control group designs use different statistical tests for dealing with statistical conclusion validity.

We will not consider the construct and external validity issues inherent in nonequivalent control group designs, for they are similar to those for experiments, case studies, and time-series designs considered above. Rather, we consider statistical conclusion validity and internal validity, in turn.

When considering the statistical conclusion validity of nonequivalent control group designs, it should be remembered that randomization is not employed. The word "nonequivalent" is used to convey the fact that one can not be sure that the populations from which the selected groups are sampled are actually the same on all pretest measures, even if there is no difference in the obtained pretest measures for the treatment and control groups. To quote Reichardt (1979, p. 148), "The label 'equivalent' does *not* imply that two groups would have *identical* mean scores on any variables measured at the pretest. Rather, it indicates that *if* the random assignment procedure were repeated over and over again so that the sample sizes in the two groups became infinitely large, the two groups would then have *identical* means (or medians, variances, or the like) on all variables measured at the pretest. Thus our use of the term *equivalent* denotes an equivalence of expected (population) values and not an equivalence of obtained (sample) values. We use the term *nonequivalent* in a similar fashion; assuming that if the same nonrandom selection process were repeated over and over again, the two treatment groups would *differ* in a number of ways" [italics his]. The bottom line is that "[w]ithout randomization, selection differences between the groups are inevitably introduced at the start of the [study]" (p. 197).

Reichardt (1979) reviews three principle analysis methods for controlling for selection differences measured by the pretest. The first method is analysis of covariance (ANCOVA). It examines the difference in the groups' posttest scores as a function of the pretest scores. A significant effect when pretest differences are statistically controlled for suggests that improved performance is a function of the treatment and not the groups' starting point. The second method is analysis of variance (ANOVA) with blocking or matching. Block membership (e.g., high versus low pretest score) is entered into the ANOVA so that one can test the effect of the pretest level, the treatment versus control group condition, and interactions between the two on the posttest scores. The third method is an ANOVA with gain

scores. That is, one performs an ANOVA on the experimental conditions, but now using the change in performance between the pretest and posttest as the dependent variable.

In his review, Reichardt (1979) points out that all three methods have threats to statistical conclusion validity. For example, random measurement error in the pretest can bias the estimate of the regression slope (used to estimate the pretest-posttest dependency) in the ANCOVA and, thereby, bias the estimate of a treatment effect. In the case of an ANOVA with matching, it is often difficult to match treatment and control representatives in actual settings because these groups tend to differ in their extreme values. For example, the treatment group may have a smaller number of high pretest scores. In an effort to "match" the groups, one might drop representatives with extreme values and, potentially, instill a systematic bias in the estimated treatment effect. And the statistical power of the ANOVA using gain scores, as compared to the ANCOVA or ANOVA with matching, depends heavily upon the particular circumstances of the research because the former tests whether the mean pretest-posttest change is significantly different between groups, not whether the mean posttest scores are significantly different. "The obvious conclusion is that none of the above techniques or any others should be blindly or thoughtlessly used to analyze data from nonequivalent group designs ... It must be remembered that a statistical technique specifies a model for the data ... Thus the statistical model must be carefully tailored to fit the unique characteristics and demands of the data at hand" (p. 186).

We now turn to consider internal validity. Specifically, Cook and Campbell (1979) point out that *the nonequivalent control group design controls for all but four threats to internal validity. The threats are all a function of the selection bias built into the design by the lack of randomization.* First, the design does not control for the effects of a "selection-maturation" bias because the respondents in one group might naturally change over time irrespective of the treatment. Experience is one way this might happen that is of particular importance when testing and evaluating expert systems. For example, let's assume that an expert system was fielded for use by operational personnel with less experience than experts, which were selected as the control. In this case, one would expect the novice group to improve in

performance over time as it gained experience (i.e., on the job training), irrespective of the hypothesized advantages of the expert system. Consequently, another group of novices, but now not receiving the expert system, would be required to determine whether improved performance was due to the expert system or experience.

The second threat is "instrumentation." As Cook and Campbell (1979, p. 105) point out, "It is not clear with many scales that the intervals are equal, and change is often easier to detect at some points on a scale than others. Scaling problems are presumably more acute the greater the nonequivalence of the experimental groups and the farther apart they are on the scale, especially if any of the group means approaches one end of the scale where ceiling or floor effects are likely." Such ceiling effects are quite plausible for the control group of experts considered in the example above. As a result, their performance may, simply by the nature of the measurement scale, be constrained; consequently, any positive improvement by the treatment group may, by comparison, seem significant. Conversely, if the experts were a treatment group receiving the expert system, their scores may not reflect actual performance enhancements due to the artificial constraints of the measurement scale.

The third threat to the internal validity of nonequivalent control group designs is "statistical regression" to the mean. That is, if groups are selected on the basis of extreme scores on a pretest, their scores can be expected to move back (or regress) to the average (i.e., mean) simply because of measurement error in the pretest. This might occur in the above example if members of the novice control (i.e., no expert system) group were selected on the basis of poorest performance on the pretest. Their performance on the posttest would improve due to regression to the mean, irrespective of whatever other gains were achieved due to experience.

Lastly, the fourth threat to internal validity is "local history," which represents an interaction of selection and history. This occurs when either the treatment or control group is exposed to events other than the treatment that might affect its performance. In the above example, this might occur if the expert and novice groups worked different shifts, worked in different

parts of the country (or world), were subjected to systematic differences in administrative procedures or supervision, etc.

With the discussion focusing on threats to internal validity, the reader should not lose sight of the fact that the nonequivalent control group design is a reasonably good one. It is certainly far superior to all three pre-experimental designs.

CHAPTER SUMMARY

This chapter has reviewed the use of experiments, appropriate case study designs, and two types of quasi-experimental designs for performing empirical tests and evaluations of expert system technology. The definitions for these empirical evaluation methods are presented in Table 6-4 for summary purposes. The discussion has been at a general level.

Table 6-4: Definitions of Empirical Evaluation Methods

Experiment (factorial):

> One or more factors are systematically varied as the independent variable(s); participants (or organizational units) are randomly assigned to the independent variable conditions; and the dependent variables are quantitative (and preferably, for our purposes, objective) measures of system performance.

Case Studies:

> An empirical inquiry investigating a contemporary phenomenon within its real-life context; using multiple sources of evidence; and striving to explain how or why something happened by logically linking the data to the propositions supporting one rival hypothesis versus others.

Quasi-Experimental Designs:

> Settings that permit some control over the scheduling of data collection even though one does not have complete control over the scheduling of experimental stimuli as provided by randomization. (Simple interrupted time-series designs, the time-series design with a control group, and the nonequivalent control group design were considered.)

Although some of the specifics of implementing these different empirical test and evaluation methods have been considered, there has been no attempt to cover all the details that would be found in experimental design and statistical analysis texts. Rather, the goal was to sensitize the reader to the different issues that should be considered when attempting to empirically demonstrate the performance benefits of an expert system. For that reason, the four types of validity, which are summarized in Table 6-5, were used as criteria for considering each of the different approaches. In particular, by focusing on the threats to these different types of validity, it is hoped that readers will be able to better formulate research designs (i.e., strategies) for assessing the performance impact of an expert system. Or, to put it more bluntly, without well formulated and conducted empirical evaluations, one has no way of knowing whether the expert system helps, hinders, or has no effect on performance.

Table 6-5: Definitions of Reliability and Validity

Reliability:

Demonstrating that the operations of a study can be repeated with the same results.

Validity:

Demonstrating that the results of a study are well grounded. The different types of validity:

Internal Validity - Establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships.

Construct Validity - Having good operational measures for the concepts being measured.

Statistical Conclusion Validity - Ensuring that the study is sensitive enough to permit reasonable statements regarding the covariation between independent and dependent variables, and using appropriate statistical tests of this covariation.

External Validity - The extent to which the results of the study can be generalized to the populations, settings, treatment variables, and measurement variables of ultimate interest.

In closing this chapter, we again note the overriding perspective represented by the SHOR paradigm that has guided this presentation. Specifically, we see the evaluator's job as helping members of the sponsoring team decide whether the expert system is an effective option for dealing with hypotheses regarding the current and/or future problem environment with which the organization will be dealing. Remember, the initial decision to build the expert system was nothing more than a hypothesis that the expert system will improve the organization's decision making and, in turn, its performance. Evaluators can use empirical test and evaluation methods to not only help members of the sponsoring team assess the adequacy of this hypothesis, but also identify what corrective actions to take if the expert system does not significantly improve performance. This latter point is extremely important from a prototyping perspective, for empirical feedback provides critical guidance for improving the expert system. Correspondingly, the failure to identify whether or not the expert system actually improved performance eliminates the possibility of improving a deficient system.

CHAPTER 7:

PULLING IT TOGETHER

The framework described in Chapter 3, along with the detailed technical, empirical, and subjective method described in Chapters 3 through 6, provides a comprehensive way to guide the testing of expert systems. A key feature of the approach, however, is the incorporation of judgments regarding the relative importance of attributes. This forms the basis to guide the testing by directing testing activities to those areas that are regarded as most important and by using testing resources most intensely in the most important areas. This chapter explains the considerations that go into these judgments and offers suggestions to the tester who is faced with the task of determining what to test and how to aggregate the results of the tests. This chapter, unlike the previous ones, contains many opinions of the authors that have not been subjected to extensive research or application. These opinions are offered in the spirit of useful suggestions rather than definitive fact. The further use of the methods described in this handbook will contribute refinements to these suggestions.

A key aspect in "pulling it all together" is a four-step approach to using the framework in Figure 7-1 (which is a reproduction of Figure 3-9). First, establish the relative importance of the different major areas (the top level in the framework: knowledge base, inference engine, "service," performance, and usability), then sub-areas, and then attributes. This information is then refined into weights. Second, examine each attribute, determine its measure, and determine how to collect that information. Third, collect that information about the system being tested. Fourth, process the information through the MAUA. Fifth, evaluate the results by comparisons between actual results and the desired or required results.

WEIGHTING DIFFERENT PARTS OF THE HIERARCHY

A useful way to establish weights in the hierarchy of Figure 7-1 is to start at the top and determine the relative importance of each major area: knowledge base, inference engine, "service," performance, and usability. The answer might be that usability and knowledge base are equally most

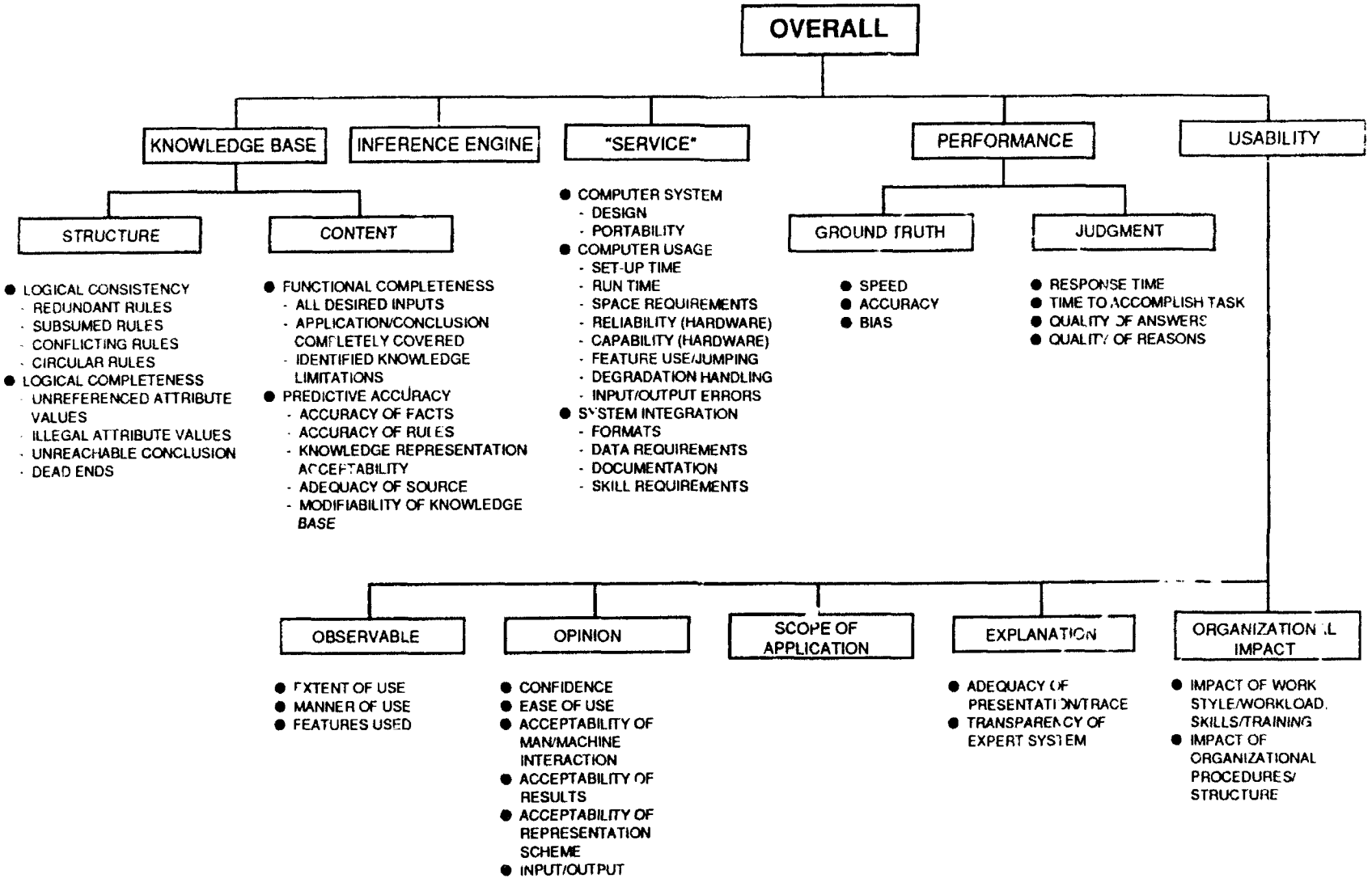


Figure 7-1: A MAU Framework for Testing and Evaluating Expert Systems

important, followed by performance which is half as important as usability, and service which is half as important as performance. Inference engine might be regarded as unimportant for testing because the developer's testing is considered sufficient assurance of the quality of the inference engine. These judgments yield the following set of weights:

Attribute	Weight	Normalized Weight (rounded)
Knowledge base	1	.364
Inference engine	0	0
Service	.25	.091
Performance	.5	.182
Usability	1	.364
Total	2.75	1.00

The numbers in the column headed "Weight" are assigned arbitrarily to agree with the judgments of relative importance. The normalized weights are calculated by dividing each weight by the sum of weights. Normalized weights put the assessments on a consistent scale. This process is repeated at each level in the hierarchy. For example, within knowledge base, an assessment is made between structure and content. Within structure, an assessment is made between logical consistency and logical completeness. Within logical completeness, an assessment is made among the four factors. When this process is complete, sufficient information is available to calculate the normalized cumulative weights used in Equation 3-1. (A detailed example of the method is presented in Volume 5, the *User's Manual for TESTER_C*, the computer program that incorporates the MAU hierarchy.)

These are the mechanics of the weighing system, but where do the judgments come from? Ideally, this information will be stated in a specification of requirements for the expert system. However, the ideal is unlikely to be achieved. Many expert systems developments fail to produce any documentation of requirements, much less a specification of the relative importance of testing different aspects of the system. In the absence of a requirements statement, the program or project manager is the best source of information about the relative importance of attributes. It is unlikely that the manager will answer the full set of questions to specify weights for the complete hierarchy, but it should be possible to elicit tradeoffs at the top level or two of the hierarchy. The tester may then need to use his own judgments,

based on whatever information is available to get the rest of the weights. Fortunately, the top weights are the ones that have the greatest effect on what actually needs to be done to test the system. For example, high importance on usability indicates that the tester should try to get human subjects to use the system in a realistic setting and fill out questionnaires. Conversely, high importance on knowledge base indicates that extensive static and dynamic testing of the code may be needed. High importance on performance indicates a need for a good set of test cases. If the program or project manager is unavailable or otherwise unwilling to provide these judgments, then the program's developer may have some insights to the tradeoffs.

If no other source of tradeoff information is available, then the tester must use his or her own judgment. This is a tricky prospect, and it is always a good idea to generate a tentative set of tradeoffs and then attempt to confirm or disconfirm those tradeoffs with a more authoritative source. In any case, the source of tradeoff information and its authority should be stated clearly in any test report.

In the absence of a more authoritative source for tradeoff information, we offer the following suggestions based on a characterization of the expert system and its intended use. This guidance is the opinion of the authors and is not intended to replace more authoritative sources. It is offered as an aid to testers who would otherwise be unable to continue without tradeoff information. It is also offered as a suggested starting point to inform a tester prior to his interviewing other sources such as the sponsoring agency.

To use our suggestions, it is first necessary to characterize four aspects of the expert system: mission criticality, degree of automation, expertise of the user, and the degree of distribution of the system. A mission critical system is one that can affect seriously the health and safety of people or large amounts of resources. It might be described compactly as a system whose errors or misuse could result in loss of life or in large dollar losses. Degree of automation refers to the extent to which the system makes decisions on its own rather than provides advice to a human operator who makes the decision. Expertise of the user: Is the user an expert or a novice? If a novice, is the user expected to increase in expertise by using the system? The degree of distribution refers to the extent to which the system is

intended for use outside of the development group. (Notice that these characteristics are not mutually exclusive.)

Mission Critical System

The first characterization is whether the system is mission critical. If a system is mission critical, then this is an overriding consideration for testing. For mission critical systems, it is extremely important to establish a minimum competency for the expert system to ensure that it will not make catastrophic errors. Important criteria requiring the greatest testing attention are:

1. *Knowledge Base and Inference Engine.* Testing is needed to ensure that the expert system will not produce disastrous decisions or advice.

2. *Quality of Answers.* Here, testing should establish whether the answers are an improvement over what would be done in the absence of an expert system. Performance against ground truth is also important.

3. *Computer System and Speed.* If specifications exist, these should be tested, probably against a pass/fail criterion.

4. *Inference Engine.* Testing should be aimed at determining that a disaster will not be caused by a faulty inference engine.

Other characteristics, including usability, are not as important for testing. If an expert system is mission critical, then intended users should be trained to overcome usability problems. If any testing is to be done on usability, it should aim at identifying areas where further development or training might be required. There may be exceptions to this general guidance, cases where the tester must test a mission critical system for usability. In such cases, it is imperative that the test subjects replicate closely the intended user. This includes replicating the training that the typical user would undergo. It also includes replicating the user's rank. If a mission critical expert system will be used by a general and if it must be tested for usability, the general must be used as a test subject.

Automatic System

Expert systems that operate automatically without human interaction require a higher degree of accuracy, since there is no chance for a human to override and correct an erroneous decision by the system. By the same token, usability factors are unimportant, since there is no human user, but system integration features (analogous to usability features for a machine) are important. The following are the most important criteria:

1. *Performance*, especially against ground truth.

2. *System Integration*, especially if the expert system is embedded in a larger system.

3. *Knowledge Base*:

   a. Accuracy of answers and facts. If the system operates automatically, there is no chance for a human to correct a system error.

   b. Functional completeness. The system must be able to reliably address all the interactions for which it is designed, since a human cannot intervene.

   c. Logical consistency. To the extent that inconsistencies can cause erroneous performance, i.e., conflicting or circular rules.

4. *Judgmentally Assessed Quality*. For areas where judgment is the only way to assess the quality of the answer, this is important (again, because a human will not be able to correct a mistake by the system).

Unless the use of the automatic expert system will impact organizational factors, all usability criteria are unimportant.

Expertise of the User

Assist an Expert. Many expert systems are designed to assist a user who is already an expert. For systems that are not mission critical and that assist an expert, greatest testing attention should address:

1. *Acceptability*. The system must be acceptable to the expert or it will not be used. If the system has to support multiple experts

in an organization, then it must be acceptable to the organization.

2. *Ease of Use.* It is important that the expert finds the system easy to use.

3. *Judgmental Quality of Answers.* For the expert to rely on and use the system, he must feel confident about the quality of answers that this system produces. Nonjudgmentally determined quality indicators are less likely to impress the expert.

4. *Organizational Impact.* If the system is developed somewhere other than at the intended user's organization, its organizational impact is an important determinant of whether or not the system will be used.

Since the person assisted is an expert, logical problems (consistency, completeness, and accuracy) are much less important and require less testing.

Assist a Novice. Many expert systems are intended to assist a novice and to improve a novice's performance, possibly to an expert's level. For systems that are not mission critical and that assist a novice, the following should receive greatest attention in testing:

1. *Judgmentally Determined Quality of Answers.* This may provide the novice with experiences similar to having an expert mentor. The novice is assisted with answers and reasons that are similar to those of an expert.

2. *Quality of the Knowledge Base.* The novice is less able than an expert to compensate for deficiencies in the structure or content of the knowledge base.

3. *Usability.* Increased usability will likely translate into increased use by the novice. However, usability is less important for a system that assists a novice than for one that assists an expert because it is assumed that a novice will have a higher tolerance for training than will an expert.

4. *Extent of Coverage.* Again, the novice cannot compensate for deficient coverage.

5. *Organizational Impact.* Organizational impact will influence heavily the extent of use.

Widely Distributed

Some expert systems are developed with the intention of being used widely; i.e., the system will be used by people who were not involved in the development and who may even be in different organizations. Such systems may also cause a large organizational impact by changing the way that things are done. Such systems are the most difficult to test and, except for mission critical systems, the most important to test extensively and well. All factors need to be considered in testing widely distributed systems, because the extent and manner of use cannot be predicted precisely. Different uses and users will likely encounter different types of problems which could lead to errors or disuse. To the extent that a widely distributed system is also one of the types mentioned above, more attention should be given to some attributes, but all attributes need to be tested at some level, and minimal thresholds of performance need to be met on all attributes (although the threshold may be very low for some attributes for some situations).

Two other characteristics of the expert system setting may also be important—a constrained computer environment and a tight testing budget. Some expert systems will operate near the limits of its computer's environment due to either hardware or software considerations. (The condition may manifest itself by slow operations.) In a constrained environment, the tester should pay some attention to factors that affect performance in the constrained environment. These include computer usage factors and factors that affect efficiency—redundant and subsumed rules, unreferenced attribute values and dead ends, and features used. Examining these attributes enables the tester to suggest changes to the system (e.g., to delete a little-used feature) that could relieve some of the environmental constraints and improve performance or other factors (e.g., speed).

Sometimes, extremely limited resources are available for testing. In this case, what should a tester do? First, if the system is mission critical, the tester should inform the appropriate manager of the inadequacy of the testing resources and the possible dire consequences of using the inadequately tested system. If no additional resources are forthcoming, the tester could test those attributes listed above to the extent possible. Any testing report should clearly state the limited scope of the tests and opinions of its

inadequacy. For systems that are not mission critical, limited testing may suffice. Systems should be subjected to several cursory overall checks by running a few test cases (e.g., a few typical cases and a few extreme cases that are most likely to cause problems). Then, areas of highest importance (e.g., as outlined above) should be tested, possibly using some of the less resource-intensive methods. Test reports should again state the limited scope of testing and qualify any conclusions as appropriate.

PERFORMING THE TESTS

After weights are specified at all levels in the hierarchy, the expert system is tested against the bottom-level attributes. These tests provide the information needed to apply Equation 3-1 and to complete the MAUA as described in Chapter 3. Testing may be conducted by applying a combination of the subjective, technical, and empirical methods described in Chapters 3 through 6. It may not always be possible to test the expert system against every criterion in a thorough manner. This section provides some informal advice on the factors that most affect the ability to test a system against the various criteria, a description of an extensive test, and suggestions for how to perform a reduced test. This advice is arranged according to sections in the hierarchy. Again, the advice is the opinion of the authors.

Knowledge-Base Structure

Factors Affecting the Ability to Test.

1. *Time.* A complete testing of the knowledge-base structure for consistency is time-consuming, especially in the absence of automated tools.

2. *Testing Resources.* Because of the time needed, it is expensive to perform this type of testing.

3. *Automated Tools.* Testing would be facilitated by automated tools to characterize the knowledge structure and check it for undesirable properties. However, such tools are not now readily available.

Testing for Logical Consistency. To test extensively for logical consistency, a tester must:

1. Characterize the entire knowledge structure in an appropriate representation scheme such as a directed graph or matrix.

2. Examine the structure for undesirable characteristics such as redundant, subsumed, conflicting, and circular rules.

For other than rule-based systems, similar tests are possible if the knowledge representation scheme (e.g., frames) can be transformed into an equivalent rule base. A thorough examination of the logical consistency is impossible for a large expert system.

In cases where exhaustive testing is impossible or impractical (e.g., in large systems), a sampling of the knowledge base may be subjected to testing. One sampling scheme that is often used is to check rules in pairs only. This will detect many of the most common problems of consistency, but will not detect the more complicated consistency errors introduced in longer logic strings (see Chapter 4 for an example). Checking pairs of rules is a common testing strategy and about the only one discussed in the literature.

Another strategy is to test a sampling of the knowledge base. This could use techniques of simple random sampling or stratified random sampling. Where it is possible, a stratified sample should be used that contains more instances of critical rules—that is, rules that could lead to those decisions with the potential for large errors.

Testing for Logical Completeness. In general, characteristics of logical completeness affect the efficiency of operation of the expert system. This, in turn, may affect the timeliness of response and operations. Characteristics of logical completeness influence errors less directly than do characteristics of logical consistency. However, in some situations, the inference engine may designate a different order of rule firing when the system is stressed, and this could lead to problems in logic. In addition, errors of logical completeness are indicative of sloppy programming, which may indicate other problems.

Extensive testing for errors of logical completeness uses the same techniques as extensive tests for logical consistency. Reduced testing for errors of logical completeness uses the same techniques as does reduced

testing for logical consistency. In addition, the data for some tests of logical completeness are generated as by-products from other tests. In particular, test cases that are developed to test the quality or acceptability of answers indirectly provide data for tests of logical completeness. For example, cases where the expert system fails to provide an answer to a valid set of inputs could be due to errors in logical completeness. Upon noticing such a problem, the tester could follow up with a more detailed investigation into the possibility of this type of logical error.

Knowledge-Base Content

Factors Affecting the Ability to Test. Unlike characteristics of logical consistency and completeness, which can be assessed strictly by the logic in the knowledge base, an assessment of the content of a knowledge base requires an external standard. There are several possibilities for providing such a standard.

1. *Requirements Document.* A complete requirements document should sufficiently describe the area of application so that these features can be tested.

2. *Access to Expert.* In cases where requirements are insufficiently documented, an expert may be able to provide the necessary description of the application.

3. *Access to the Operating Setting.* In cases where requirements are insufficiently documented and an expert cannot adequately describe the functional requirements, the tester may be able to infer these requirements from an examination of actual input data and observation of people performing the tasks that the system will perform or aid.

The accuracy of rules, especially for a knowledge-based system, might be determined from an expert or panel of experts, and these experts must be available. (An important exception is where some part of the rule base encodes physical laws, in which case an expert is not needed.) An expert is also needed to determine the adequacy of the knowledge representation. Because of the expense of using experts, sufficient resources are needed to test the content.

Testing for Functional Completeness. The following activities should be included in an extensive test for functional completeness:

1. Determine the scope of application. If a complete requirements document exists, this will provide a description of the range of application including all types and numbers of inputs, all applications, and the range of conclusions that should be possible.

2. Confirm the scope of application. By examining actual data and observing as much of the decision setting as possible, confirm the range of inputs, applications, and range of conclusions required. If there are differences between the observed scope and the stated scope, make a note of the differences in the test report.

3. Test that the system responds to the range of data with the range of conclusions. Exercise the system with test cases to determine if the system responds appropriately. The choice of test cases should include: typical or representative data (both quality and quantity), data representing extreme inputs, and cases expected to return extreme conclusions (especially more important conclusions). If previous sets of test cases (e.g., those used in development) are available, run these as well for a regression test. Representative test cases provide information on normal operations of the system. (By keeping track of the system's operating characteristics such as speed, this test will also provide information in assessing other attributes.) Extreme test cases may be more important for assessing functional completeness. The system should provide appropriate conclusions even to extreme inputs. Similarly, legal inputs should be all that are required to support extreme conclusions that are within the scope of application. (This may be more easily tested in backward-chaining systems.)

4. Select test cases outside of the range of application. Both inputs and conclusions outside of the range of application should be tested. This will provide data to assess whether the system knows its limits and responds appropriately.

There are two areas that might cause testing for functional completeness to be reduced—determination of the scope of application and selection of test cases. In cases where a requirements document is not available, experts are not available, and the decision setting is inaccessible to the tester, the scope of application may need to be inferred by examining the code or by observing the system's interactions and output. This will require more of the tester's effort, but does not require resources that are not available. If this method is used to infer the scope of the application, the tester should be sure to state this clearly in the test report.

The other area for reducing testing is in the selection of test cases. A reduced set may require fewer resources to select, run, and analyze. Even with a reduced set, the tester should attempt to give some coverage to extreme and important cases and some attention to normal operating cases.

Testing for Predictive Accuracy. The following activities are needed for an extensive test of predictive accuracy:

1. *Check all facts against a recognized authority.* The authority could be a reference document, a written regulation, an expert, or a panel of experts. Reference documents are generally the best sources for checking factual information. A single expert may be sufficient to verify facts, but we recommend a panel of experts to verify the accuracy of rules. In cases where the facts will change during the operation of the system (e.g., when the facts are the contents of intelligence reports), the process for receiving, verifying, and changing facts should be tested. This test will need to use a sample of test cases. Some test cases should be chosen to be representative of the actual operations; others should reflect the extremes of what might happen.

2. *Check all rules.* Check all rule sequences against the consensus authority of a panel of experts. In most actual expert systems, the number of sequences of rules will be too large to check and a sampling is needed. If so, the sample should contain sequences that represent normal operations, unusual sequences, and other extreme sequences that could lead to extreme conclusions.

3. *Verify the adequacy of sources.* Official guidance or regulations may specify some sources of facts. At the other extreme, a consensus of experts may be needed to certify the expert whose expertise was encoded in the system.

4. *Test the procedure for changing the knowledge base.* Determine whether, and under what conditions, the knowledge base (e.g., rules, frames, etc.) could be changed, who is allowed to make the change, and how. Assess the criticality of the changeable portion of the system, and verify that allowed changes are appropriate. The authority on the appropriateness of the changes could be a regulation, a requirements document, an expert, or a manager of the operation.

Several changes in procedure could be used if resources or conditions do not permit or require extensive testing of content. These include:

1. *Use a single expert instead of a panel of experts.* It is usually cheaper and easier to use a single expert, and, unless the system is critical, a single expert will often suffice.

2. *If requirements documents do not address the factors that the user should be allowed to modify, the tester could use his judgment.* If the tester uses his judgment, he should state this, along with any reasoning, in the test report.

3. *Use available documents in the absence of authoritative references.* In these cases especially, the sources should be indicated in the test report.

4. *Rely on reputation or apparent expertise (e.g., as indicated by position or title) to assess the adequacy of the source.* If it is difficult to establish credentials definitively, this may suffice.

Inference Engine

As a practical matter, it is difficult to test an inference engine, and most testers do not even try. For noncritical applications of widely used and established environments, tools, or shells, this practice should not cause a serious problem. The widespread use of the tool will probably turn up most of the problems with the inference engine, and the noncritical nature of the application limits the seriousness of possible problems. Furthermore, other tests (e.g., those aimed at discovering the correctness of reasoning or of conclusions) can find some of the problems that could be caused by a faulty inference engine.

The development of benchmarks would aid in the testing of inference engines. A benchmark is a standard module of coded knowledge with known, proven-correct results that can be coded on a variety of inference engines. The correct performance of the inference engine on a comprehensive set of benchmarks provides strong evidence that the inference engine is correct. Unfortunately, such a set of benchmarks has not yet been developed.

Another approach to testing inference engines is to code identical knowledge bases in different inference-engine products (e.g., shells) and exercise these programs in parallel. If the behavior of all of the systems is the same, this gives some evidence that the inference engines are free of problems. However, this evidence is not absolutely conclusive. Furthermore, different results indicate problems in one or more inference engines, but may not indicate which particular inference engine is faulty. This procedure is

also expensive, and the expense will be hard to justify for any but the most critical expert systems.

"Service"

Factors Affecting the Ability to Test.

1. *System Description.* In order to test system compatibility and integration, the target hardware, software, and organizational systems need to be specified and defined.

2. *Automated Tools.* Testing for internal machine characteristics may require automated support tools.

3. *Access to Hardware and Software.* Expert systems that are designed to be operated on several systems should be tested on those systems, and this requires access to the full range of intended hardware and software.

Testing for Service Requirements. An extensive test for service requirements would include the following:

1. Direct measurement of some items such as set-up time.

2. Use of internal mechanisms or automated tools such as internal "clocks" and internal machine utilization maps.

3. Operating the expert system on the full range of targeted hardware and software platforms.

4. Complete checking of system integration attributes.

A reduced test of service attributes could be attained by attempting to operate the expert system in a reasonable approximation of its intended operating setting (hardware, software, operating personnel, organization) and noting problems with "service" attributes.

Performance

Factors Affecting the Ability to Test.

1. *Ground Truth.* Actual test cases with correct, ground truth answers form a solid basis for testing performance. Most expert

system applications are not directed at situations where such answers exist. However, it is still useful to search for parts of an expert system (e.g., a forecasting module) that can be tested against ground truth.

2. *Availability of Experts.* Experts are likely to be required to judge the performance of an expert system. Experts are generally scarce and expensive.

Testing for Performance. To test an expert system's performance thoroughly, a test should:

1. *Use many test cases.* These cases should include instances of ground truth, as well as instances that require expert judgment. Cases should be chosen to simulate the expected and extreme aspects of the intended use of the expert system, both in the difficulty of the problem and the level and timing of input. In addition, some cases should probe the limits of the system's operation, by being at the extremes of inputs and at the extremes of consequences.

2. *Use a group of experts.* We recommend the use of a group of experts to judge the quality of the expert system's answers and reasons.

3. *Analyze data and perform statistical tests.* Chapter 5 provides the details on how to analyze data.

Reduced testing of performance would use fewer test cases and fewer experts. Test cases should still be chosen to represent the expected situation and some of the extremes. The methods described in Chapter 5 are useful for choosing sample sizes. A single expert will often suffice if the expert system is not mission critical.

Usability

Factors Affecting the Ability to Test.

1. *Availability of Subjects.* Subjects are necessary to test for usability. Usability features can be ascertained only by observing or questioning users.

2. *Access to Realistic Circumstances.* Having actual hardware, software, other equipment, realistic problem scenarios, and realistic personnel and organizational setting provides the basis for a reliable test of usability.

3. *Special Facilities.* Special facilities such as unobtrusive, one-way observation rooms aid in testing observable aspects of usability.

Testing for Usability. An extensive test for usability will include:

1. *Use many test subjects.* Ideally, the test subjects will have the level of skill and training, including training in the use of the expert system, as the ultimate intended user. The use of many subjects will allow the tester to perform statistical analyses of usability data.

2. *Use realistic settings.* Ideally, the organizational setting, as well as hardware, software, and other equipment, should replicate the setting in which the expert system will be used. Additionally, a full range of realistic test scenarios should provide the basis for assessing usability under extreme conditions such as time stress, as well as under routine conditions.

3. *Administer questionnaires.* Both types of questionnaires in the Appendix are appropriate for eliciting usability information. In an extensive test, both questionnaires should be used and this should be supplemented by open-ended interviews and follow-up questions.

4. *Observe behavior.* The behavior of subjects should be observed and the extent of use, manner of use, and features used should be recorded. This information can counter biases that subjects may express in answering questionnaires. Ideally, an unobtrusive one-way observation room should be used so that the observation does not affect the subject's behavior. Some aspects, such as a log of features used might be automated as part of the computer system, but care should be taken to avoid changing the operation of the expert system.

Reduced testing for usability can be achieved by using fewer subjects, fewer scenarios, and fewer questioning techniques. Even with fewer subjects, attempts should be made to obtain subjects with the appropriate skill and training levels. In any case, the tester should describe the test subjects in the test report (e.g., by number, rank, training, job function, etc.). Fewer scenarios should still reflect both the normal operating condition and a stressed condition, if possible. In many cases, the simpler questionnaire (the one with Likert-type scales) should be sufficient. Extensive observation may be impossible, but even a cursory observation by walking around near the end of the test can provide some useful information.

CHAPTER SUMMARY

This chapter addressed aspects of pulling the ideas from Chapters 3 through 6 together in a test of an expert system. We first discussed aspects involved in establishing weights in a multiattribute utility framework and how this framework can be used to elicit information about important things to test from sponsoring agencies. Next, we offered suggestions on the relative importance of different attributes based on a characterization of the expert system. Finally, we offered suggestions for testing activities in each major attribute category based on the extent of testing that is feasible. We also discussed factors that could influence or limit the ability to test for each category of attributes. The suggestions offered in this chapter are the opinions of the authors.

CHAPTER 8:

OTHER APPROACHES TO TEST AND EVALUATION

Thus far, we have not explicitly focused on the issues of verification versus validation, static versus dynamic testing, software quality metrics, or design and coding standards. This chapter will do so in three subsections, respectively. Each subsection will first define the issues and terms, and then identify test and evaluation criteria and attributes in Table 2-1 that address them.

As indicated in previous chapters, subjective, technical, and empirical test and evaluation methods are designed for assessing the system's score on different criteria and attributes. The criteria and attributes are the critical reference points, not the methods. By focusing on the criteria and attributes, one can subsequently identify the test and evaluation methods addressing verification versus validation, static versus dynamic testing, and various software quality factors. As mentioned earlier, we do not address software design and coding standards.

VERIFICATION VERSUS VALIDATION

Verification refers to assessing how well the system was built. Validation refers to assessing whether or not the right system was built. Verification addresses the internal correctness of the system. Validation addresses the external correctness of the system. Verification tests for logical inconsistencies in the system and programming bugs in the software. Validation tests for the accuracy of the information in the system, and for its usability by operators performing their tasks.

The shaded portions of Figure 8-1 identify the criteria and attributes addressing expert system verification; the unshaded portions address expert system validation. Verification means testing for the logical consistency and logical completeness of the expert system's knowledge base, inference engine, and "service features." The focus is on ensuring the internal correctness of the system. Even if the knowledge base's content were of

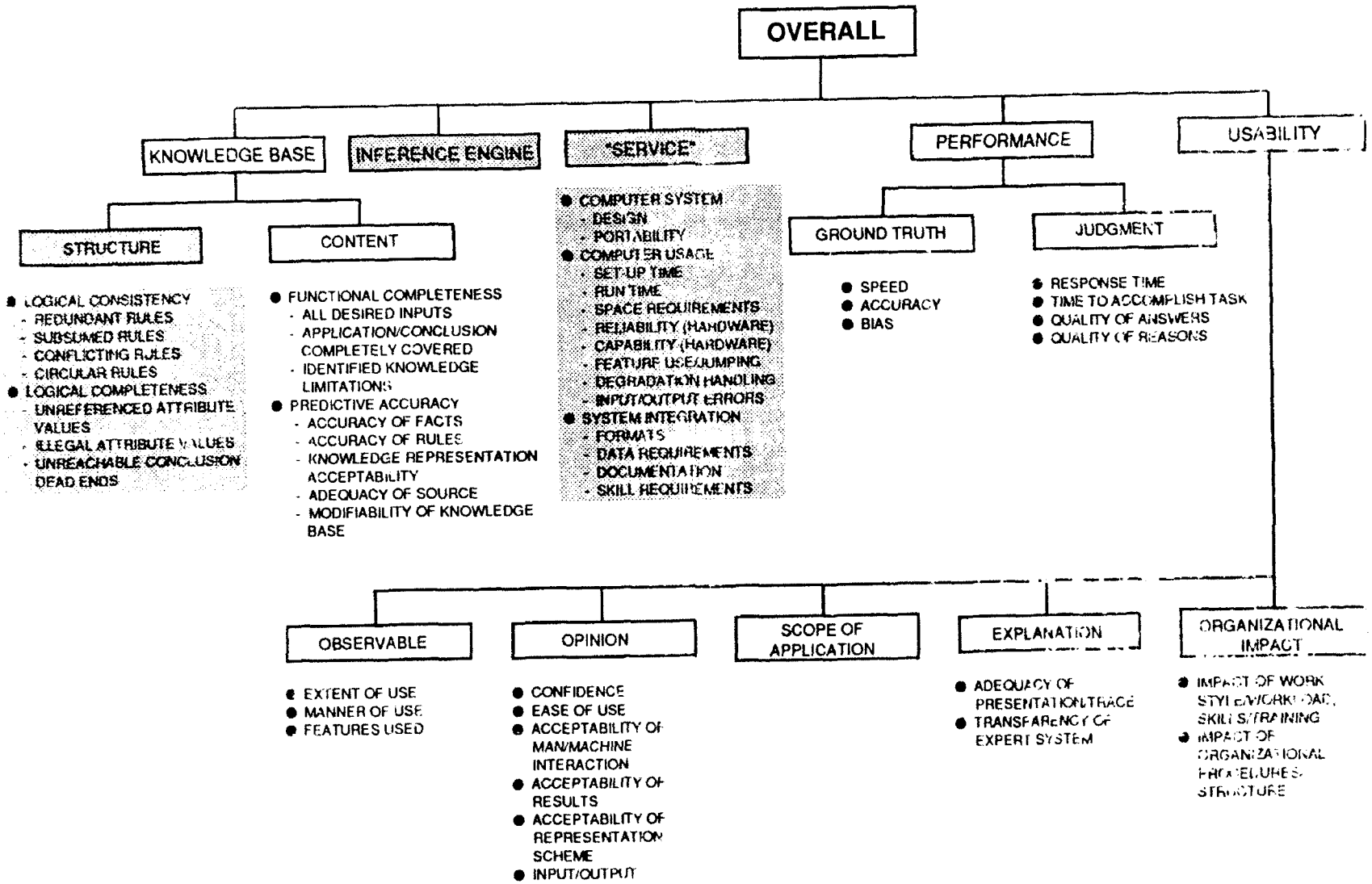


Figure 8-1: Test and Evaluation Criteria Addressing Verification (Shaded) versus Validation (Unshaded) Methods

low quality, one could still verify that the expert system was free from any logical flaws and that it was designed and programmed well.

In contrast, validation means testing for the functional completeness and predictive accuracy of the knowledge base's content, and for all performance and usability attributes of operators working with the system. The focus is on ensuring the external correctness of the system. We want to validate that the knowledge base contains expertise and, more generally, that the expert system permits its users and the larger organization to do their job better.

As Table 8-1 illustrates, the overall utility of an expert system depends on successful verification and validation testing. As we have discussed earlier, the relative importance of testing all the attributes in the hierarchy—that is, complete verification and validation—depends on the environmental conditions facing the test and evaluation team. Nevertheless, there is an inherent dependency between attributes in the hierarchy. If verification testing identifies severe logical flaws in the system, users will probably not perform well regardless of the expertise embedded in the knowledge base. Similarly, if validation testing identifies severe performance and usability deficiencies with the system, they may be due to the internal correctness of the knowledge base, inference engine, or "service" components, and not the knowledge base's expertise. For these reasons, complete V&V (verification and validation) should be performed whenever possible.

STATIC VERSUS DYNAMIC TESTING

Static and dynamic testing are two approaches to testing conventional software systems that are also applicable to expert systems. Static testing refers to assessing system quality without actually executing its code. Conventional static testing methods include code inspections and walkthroughs and specially designed programs to assess logic problems, structural errors, syntactic errors, and coding and interface consistency with accepted programming standards. In contrast, dynamic testing refers to using test cases to execute the code and thereby assess system quality. Conventional dynamic testing methods include functional tests, performance

Table 8-1: Linking Software Quality Subfactors to Attributes in the Hierarchy

Software Quality Factor	Software Quality Subfactor	Attributes in Hierarchy
Correctness	Traceability	All Attributes if Use Hierarchy as a Requirements Tool
	Completeness	Knowledge Base Structure • Logical Completeness • Functional Completeness Usability: Scope of Application
	Consistency	Knowledge Base Structure • Logical Consistency "Service:" System Integration • Formats • Data Requirements • Documentation
	Error Tolerance	"Service:" Computer Usage • Degradation Handling • Input/Output Errors
Reliability	Consistency	Knowledge Base Structure • Logical Consistency "Service:" System Integration • Formats • Data Requirements • Documentation
	Error Tolerance	"Service:" Computer Usage • Reliability (Hardware) • Degradation Handling • Input/Output Errors
	Accuracy	Knowledge Base: Content • Accuracy of Facts • Accuracy of Rules • Adequacy of Source Inference Engine

Table 8-1: Linking Software Quality Subfactors to Attributes in the Hierarchy (Continued)

Software Quality Factor	Software Quality Subfactor	Attributes in Hierarchy
Reliability (cont.)	Accuracy (cont.)	Performance: Ground Truth • Accuracy • Bias Performance: Judgment • Quality of Answers • Quality of Reasons Usability: Opinion • Confidence • Acceptability of Results
	Structural Simplicity	Knowledge Base: Content • Knowledge Representation Acceptability Usability: Opinion • Acceptability of Representation Scheme Usability: Explanation • Transparency of Expert System
	Test Adequacy	All Attributes in the Hierarchy are Designed for Thorough Testing of System
Efficiency	Execution Efficiency	"Service:" Computer Usage • Run Time Performance: Ground Truth • Speed Performance: Judgment • Response Time • Time to Accomplish Task
	Storage Efficiency	"Service:" Computer Usage • Space Requirements • Capability (Hardware)

Table 8-1: Linking Software Quality Subfactors to Attributes in the Hierarchy (Continued)

Software Quality Factor	Software Quality Subfactor	Attributes in Hierarchy
Integrity	Access Control	Knowledge Base: Content • Modifiability of Knowledge Base "Service:" Computer System • Design
	Access Auditability	
Usability	Communicative-ness	Usability: Observable • Extent of Use • Manner of Use • Features Used Usability: Opinion • Ease of Use • Acceptability of Man/Machine Interaction • Input/Output Usability: Explanation • Adequacy of Presentation/Trace Usability: Organizational Impact • Work Style/Workload, Skills/Training • Procedures/Structure
	Operability	"Service:" Computer Usage • Set-Up Time • Space Requirements • Reliability (Hardware) • Capability (Hardware) • Feature Use/Jumpings • Degradation Handling • Input/Output Errors • Skill Requirements

Table 8-1: Linking Software Quality Subfactors to Attributes in the Hierarchy (Continued)

Software Quality Factor	Software Quality Subfactor	Attributes in Hierarchy
Maintainability	Consistency	Knowledge Base: Structure • Logical Consistency "Service:" System Integration • Formats • Data Requirements • Documentation • Skill Requirements
	Structural Simplicity	Knowledge Base: Content • Knowledge Representation Acceptability Usability: Opinion • Acceptability of Representation Scheme Usability: Explanation • Transparency of Expert System
	Modularity	Knowledge Base: Structure
	Self Descriptiveness	"Service:" System Integration • Documentation
	Documentation Adequacy	"Service:" System Integration • Documentation
Testability	Structural Simplicity	Knowledge Base: Content • Knowledge Representation Acceptability Usability: Opinion • Acceptability of Representation Scheme Usability: Explanation • Transparency of Expert System
	Modularity	Knowledge Base: Structure
	Instrumentation	

Table 8-1: Linking Software Quality Subfactors to Attributes in the Hierarchy (Continued)

Software Quality Factor	Software Quality Subfactor	Attributes in Hierarchy
Flexibility	Modularity	Knowledge Base: Structure
	Self Descriptiveness	"Service:" System Integration • Documentation
	Documentation Adequacy	"Service:" System Integration • Documentation
	Expandability	Knowledge Base: Content • Modifiability of Knowledge Base
Portability	Modularity	Knowledge Base: Structure
	Self Descriptiveness	"Service:" System Integration • Documentation
	Machine Independence	"Service:" Computer System • Portability
Reusability	Modularity	Knowledge Base: Structure
	Self Descriptiveness	"Service:" System Integration • Documentation
	Machine Independence	"Service:" Computer System • Portability
Inter-Operability	Modularity	Knowledge Base: Structure
	Data Commonality	"Service:" System Integration • Data Requirements
	Communications Commonality	"Service:" System Integration • Formats

tests, stress tests, and structural tests. Both static and dynamic testing methods are used to test the "service requirements" of conventional software systems.

For some of the attributes in our test and evaluation hierarchy, it is clear whether static or dynamic testing methods are most appropriate. For example, static methods are most appropriate for assessing the System Integration attributes: formats, data requirements, documentation, and skill requirements. In contrast, dynamic methods using test cases, not just loading and running the software, are most appropriate for assessing the ground truth performance measures of accuracy and bias.

However, other attributes can be assessed using either static or dynamic testing methods. For example, Chapter 4 overviewed static testing methods (e.g., using flow graphs, incident matrices, and Boolean algebra) to assess the logical consistency and completeness of the knowledge base. In addition, we overviewed a program called Validator that uses test cases to assess these attributes. Similarly, specific test cases, sometimes embedded in a larger scenario, are typically used to assess the expert system's usability. However, one could also give one a demonstration of the system to obtain usability judgments. Although the system might be executed, in the sense that the software is run, realistic test cases do not have to be part of the demonstration. This is, however, an inferior approach to assessing expert system usability.

The shaded portion of Figure 8-2 identifies those criteria and attributes that are typically assessed by static testing methods. The unshaded portion of Figure 8-2 identifies the criteria and attributes typically assessed by dynamic testing methods.

SOFTWARE QUALITY FACTORS

Sizemore (1990) has recently matched various test and evaluation methods to software quality factors (or metrics) identified in the software engineering literature, especially in the *Software Quality Engineering Handbook* and in McCall and Matsumoto (1980). In particular, Figure 8-3

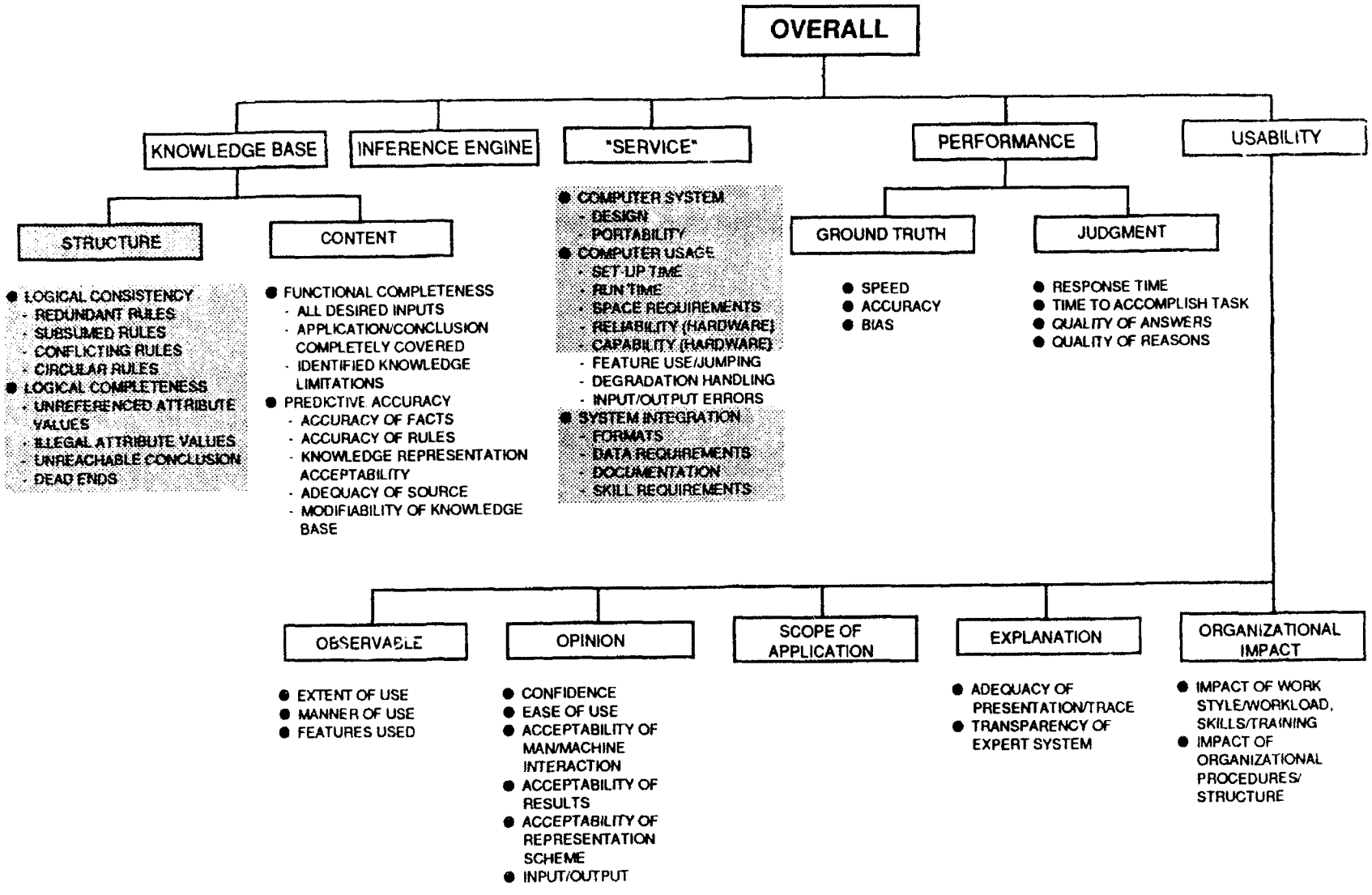


Figure 8-2: Test and Evaluation Criteria Typically Assessed by Static Testing Methods (Shaded) and Dynamic Testing Methods (Unshaded)

QUALITY FACTOR	DEFINITION	USER CONCERN	ACQUISITION CONCERN
EFFICIENCY	The ability of a software system to perform its required functions with minimum consumption of computer time and storage resources.	How well does it utilize resources?	PERFORMANCE (How well does it function?)
INTEGRITY	The ability of a software system to control unauthorized access to or modification of system software or data.	How secure is it?	
RELIABILITY	The ability of a software system to perform its required functions with correct and consistent results.	What confidence can be placed in what it does?	
USABILITY	The ability of a software system to be easily learned and used.	How easy is it to use?	
CORRECTNESS	The extent to which the software satisfies its specification and fulfills the user requirements.	How well does it conform to the requirements?	DESIGN (How valid is the design?)
MAINTAIN-ABILITY	The ability of a software system to be easily corrected when errors are discovered.	How easy is it to repair?	
TESTABILITY	The ability of a software system to be easily and thoroughly tested.	How easy is it to verify its performance?	
FLEXIBILITY	The ability of a software system to be easily modified to meet new requirements.	How easy is it to change?	ADAPTABILITY (How adaptable is it?)
INTEROPER-ABILITY	The ability of a software system to effectively exchange information with other software systems.	How easy is it to interface with another system?	
PORTABILITY	The ability of a software system to be easily modified to operate in more than one environment	How easy is it to transport?	
REUSABILITY	The ability of a software system or parts of a system to be used in multiple applications.	How easy is it to convert for use in another application?	

Source: Sizemore (1990), p.1-4

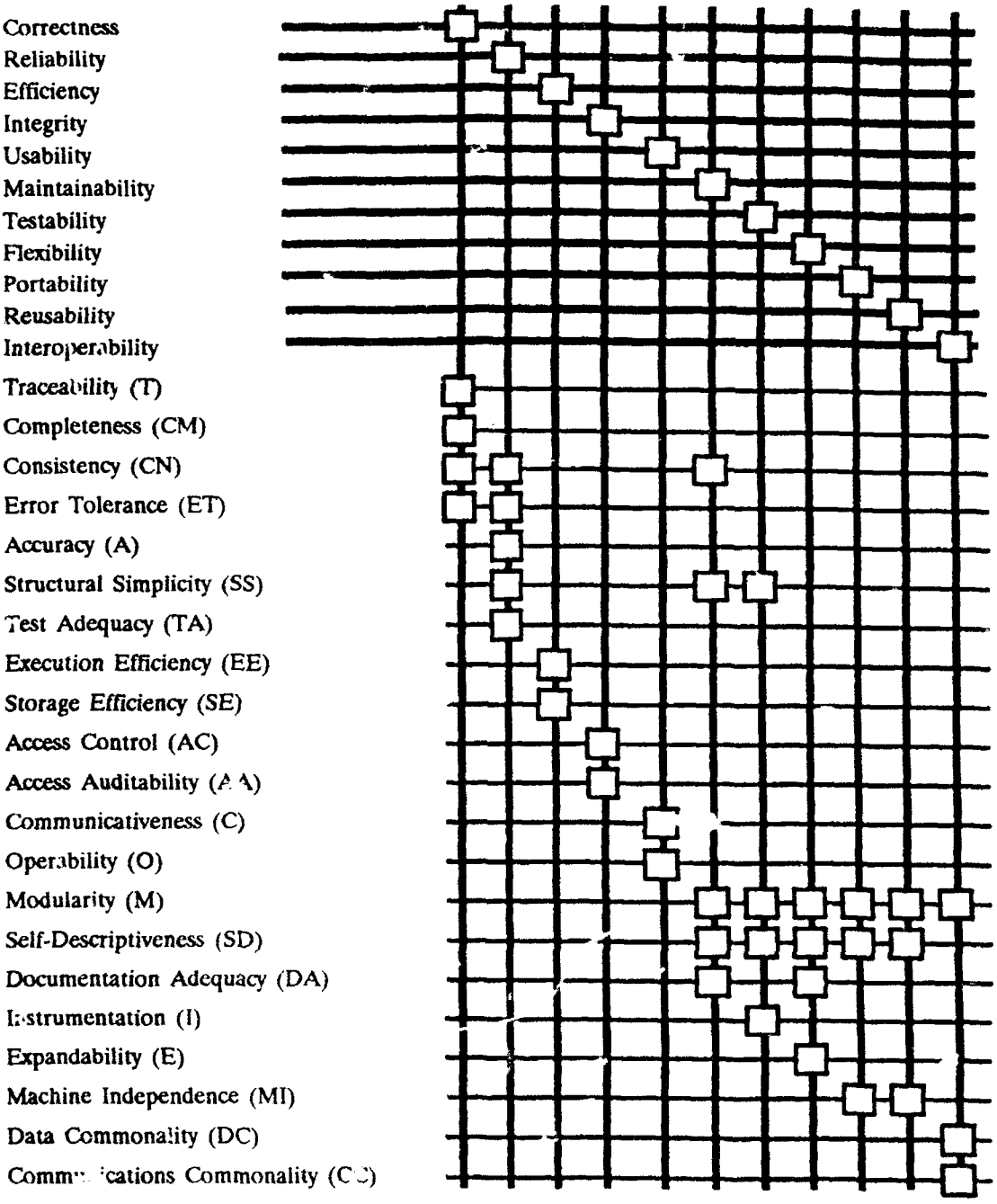
Figure 8-3: Software Quality Factors

presents the eleven software quality factors he used; Figure 8-4 presents a matrix representation of how these eleven software quality factors are decomposed into more measurable subfactors; and Figure 8-5 presents the definitions of each of the subfactors. We considered it important for completeness to assess whether the test and evaluation criteria and attributes in our hierarchy (Table 2-1 and Table 3-9) addressed the software quality factors and subfactors used in Sizemore's study.

Table 8-1 links the test and evaluation attributes in our hierarchy to Sizemore's software quality factors and subfactors. That is, for any given software quality subfactor, Table 8-1 identifies the appropriate test and evaluation attributes in our hierarchy. For example, Table 8-1 indicates the subfactor Consistency can be assessed in the hierarchy by examining (1) the logical consistency of the knowledge base and (2) four system integration attributes: format consistency, consistency with data requirements, documentation consistency, and the system's consistency with the identified skill requirements of the users. This view takes a broad interpretation of "consistency" as found by Sizemore. We suggest that much of the software engineering literature interprets "consistency" much more narrowly, especially by including consistency with skill levels of users.

As can be seen, Table 8-1 is a very long table. This occurs because many of the quality subfactors measure more than one quality factor. For example, as shown in Figure 8-3, the Consistency subfactor measures Correctness, Reliability, and Maintainability. In fact, six of the twenty-one subfactors measure more than one factor. In all cases we have repeated the subfactor to attribute linkages so that Table 8-1 provides the complete factor to subfactor to attribute linkages for the reader's examination.

We want to make four principal points. First, all the attributes in the hierarchy are linked to Sizemore's software quality subfactors. No attribute in the hierarchy is left unaccounted for. However, in some cases, the subfactor may need to be interpreted rather broadly to make the linkage. For example, the Service attribute skill requirements refer to the system's requiring an appropriate skill level from the user. We interpret this to be an aspect of operability, in that the system could not



Source: Sizemore (1990), p. 1-6

Figure 8-4: Factor to Subfactor Decomposition in Matrix Form

TRACEABILITY

The extent to which the products of each software development phase implement the products that precede them, have their basis in those products, and provide mechanisms that aid in establishing those ties

EXECUTION EFFICIENCY

The extent to which a system performs its intended functions with minimum execution time

SELF DESCRIPTIVENESS

The extent to which program source code is easy to read and understand

COMPLETENESS

The extent to which a system contains all required components and each of those components is fully developed.

STORAGE EFFICIENCY

The extent to which a system performs its intended functions with minimum consumption of storage resources

DOCUMENTATION ADEQUACY

The extent to which required documentation exists, is in the required format, and is accurate, clear, and complete

CONSISTENCY

The extent to which a system's code and documentation are uniform and free of contradiction

ACCESS CONTROL

The extent to which a system provides mechanisms to control access to software and data.

INSTRUMENTATION

The extent to which a system contains instructions or assertions to facilitate execution monitoring, debugging, and testing

ERROR TOLERANCE

The extent to which a system continues to operate correctly despite input errors or software faults.

ACCESS AUDITABILITY

The extent to which a system provides mechanisms to audit the accessing of software and data

EXPANDABILITY

The extent to which a system can be easily modified to provide additional functions or data storage capacity.

ACCURACY

The extent to which a system is free from error in calculations and output.

COMMUNICATIVENESS

The extent to which a system provides useful output and an interface with the user.

MACHINE INDEPENDENCE

The extent to which a system can be made to execute in more than one hardware or software environment

TEST ADEQUACY

The extent to which test planning and execution ensure thorough testing of the system

OPERABILITY

The extent to which a system can be loaded, initiated, executed, and terminated

DATA COMMONALITY

The extent to which a system uses standard or common data formats, types, representations, and structuring

STRUCTURAL SIMPLICITY

The extent to which a system is free from complicated data, logical, and control structures

MODULARITY

The extent to which a system is composed of discrete components such that a change to one component has a minimal impact on other components and such that the tasks performed by a single component are functionally related

COMMUNICATIONS COMMONALITY

The extent to which a system uses standard or common communication protocols and interface routines

Source: Sizemore (1990), p. 1-5

Figure 8-5. Software Quality Subfactors

be executed if its skill requirements were excessive. A more narrow interpretation could view operability as referring only to the system and if it could be executed at all.

Second, all the attributes in the hierarchy can be used to measure two subfactors: Traceability and Test Adequacy. Specifically, Traceability is addressed by all the attributes because all the attributes in the hierarchy can be used as a requirements tool, particularly in conjunction with Multi-Attribute Utility Assessment (MAUA), as explained in Chapter 3. Similarly, Test Adequacy is addressed because all the attributes need to be assessed in order to test an expert system thoroughly. All the other quality subfactors are linked to one or more, but not all, of the attributes in the hierarchy.

Third, neither the Correctness nor Testability factors have the "Accuracy" subfactor under it. Nor does Testability include the Communicativeness subfactor, which is linked to many of the Usability attributes in the hierarchy. These subfactors are probably omitted from the above factors because these software quality metrics were designed for conventional software, not expert systems. We think these factors should include the above subfactors for testing expert systems. However, in order to maintain consistency with the software quality literature, we have not included them or the corresponding attributes in Table 8-1.

Fourth, the only factors or subfactors that are unaccounted for in the hierarchy are those, such as Modularity and Self-Descriptiveness and some aspects of Maintainability, that relate to design and coding standards or those, such as Access Auditability and Instrumentation, that relate to built-in testability (which is also a design issue). As mentioned in Chapter 3, we do not address design and coding standards. This said, if we interpret these subfactors broadly, we can still find a linkage to the hierarchy, and this is done in Table 8-1.

Figure 8-6 shows the linkage in the other direction. The attribute hierarchy is shown with labels for subfactors next to each attribute.

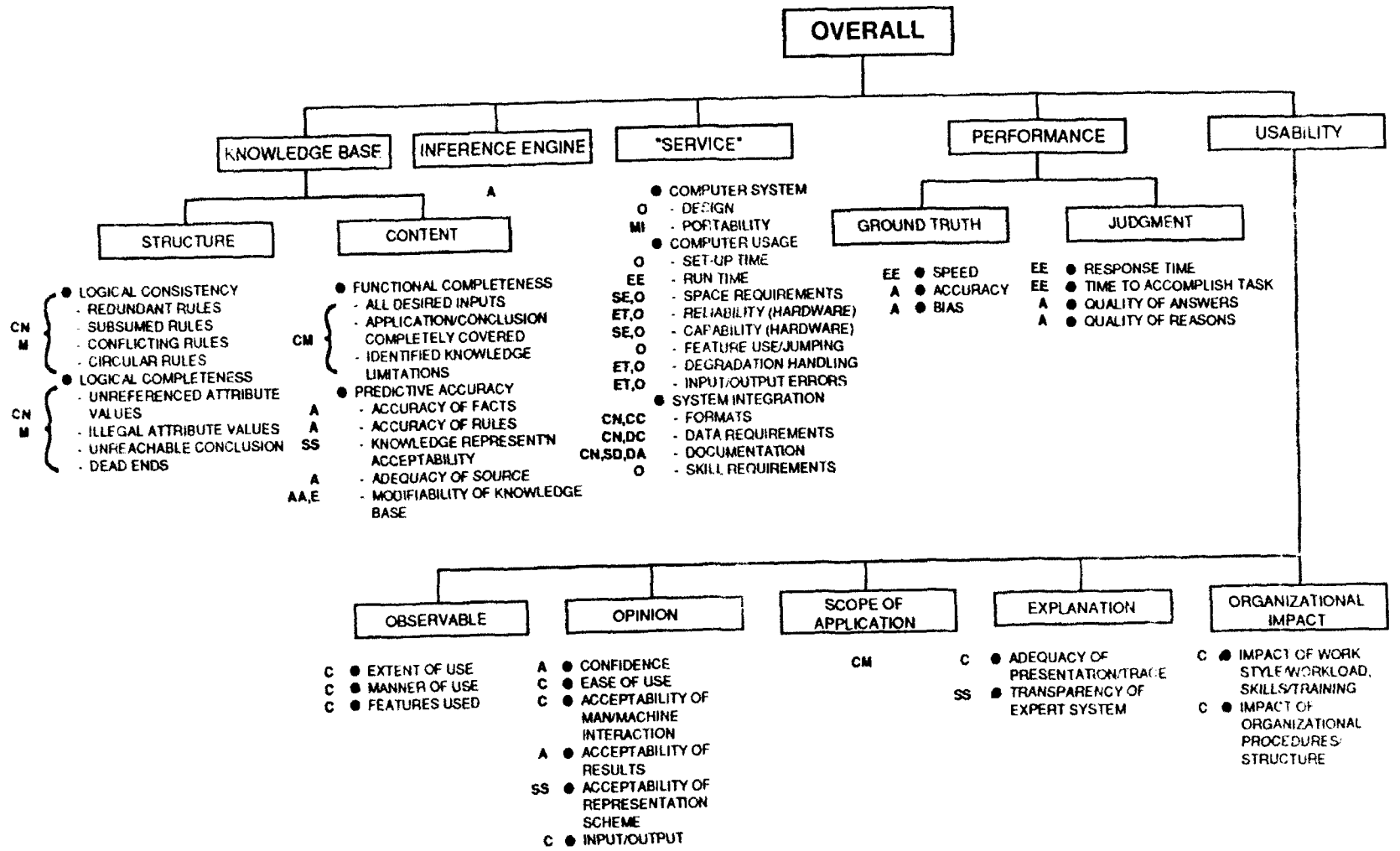


Figure 8-6: The Software Quality Subfactors Addressed by each Evaluation Attribute in the Hierarchy [Note: Does not include the subfactors "Traceability or Test Adequacy," which were linked to all attributes.]

CHAPTER SUMMARY

This chapter relates the methods developed in this Handbook to other approaches to software testing and evaluation. In particular, we show how the attributes in our framework for testing and evaluating expert systems relate to: verification and validation, static and dynamic testing, and especially software quality factors. We find that: 1) all attributes in the framework relate to software quality subfactors; 2) all attributes in the framework can be used to measure traceability and test adequacy; 3) the only software quality factors or subfactors that are not in our framework are those that relate to design and coding standards or to built-in testability.

CHAPTER 9:

FUTURE DIRECTIONS

The last few years have seen an explosion of interest in testing artificial intelligence and knowledge-based systems. As one indication, over 100 papers have appeared on the topic since 1987. However, there is still a long way to go for the testing of artificial intelligence (AI) and knowledge-based systems to reach the level of conventional software testing. This chapter sketches the components of a test technology program, which could substantially advance the science and practice of testing AI, and presents some specific suggestions that could be pursued now.

TEST TECHNOLOGY PROGRAM

A test technology program would need to address both the "science" of testing, by further developing the basis for testing, and the "engineering" of testing, by developing specific methods and tools for testing. We identify six specific items.

1. *Assess and codify the state-of-the-art in testing.* The AI testing community needs to continue its efforts to get its arms around all the diverse AI testing activities. This requires a comparison and contrast of activities (a) *within* a particular activity area (e.g., various static testing approaches), and (b) *across* areas (e.g., static testing vs. dynamic testing vs. use of experts vs. experiments, etc.). We can easily imagine task forces within and across activity areas, with the result being a major reference work in the field for years to come.

2. *Develop AI testing laboratories.* There need to be empirical evaluations of alternative testing approaches (and products) within and across activity areas, as well as of completed expert systems and expert system shells in order to assess their adequacy. The first step is to set up government or commercial laboratories (with no vested interests, e.g., not at product vendors) whose mission is to perform such empirical evaluations. It may be most cost-efficient to have different laboratories specialize in different areas, although this may be premature at this point.

3. *Package AI testing approaches and products for Army personnel.* A significant effort is required to transfer AI testing technology to Army personnel, and that effort needs to be managed carefully since items 1 and 2 above have yet to be performed. Elements of this include a training program with courses and a place where

testers can get hands-on experience, computerized support, and texts.

4. *Direct efforts toward assessing the value of integrating AI testing into the development process.* It is often argued that such integration will result in better AI systems and reduced development costs, but we are not aware of any empirical studies testing this hypothesis. This could be the first step of a larger project to get testers involved earlier in the development process to ensure that things such as requirements documents are produced to aid in testing. Managers have to be shown that this involvement is worthwhile, however.

5. *Direct efforts toward assessing the relative effect of knowledge elicitation techniques, domain experts, knowledge engineers, representation schemes, and problem domains on knowledge-base quality.* It seems quite appropriate for the AI testing community to evaluate the adequacy of the methods that go into building an expert system, not just the finished products (i.e., systems). This would be a major undertaking but especially important if AI is here to stay (e.g., see Adelman, 1989). This would also include the development of testing techniques for "funny logics" (4-valued, non-monotonic, possible-worlds, probabilistic) where appropriate tests do not always exist. Present techniques are focused primarily on rule-based systems (possibly with extensions to frames) and techniques may be needed for other types of systems.

6. *Develop testing tools.* The five items above are directed at the "science" of testing AI. This item is directed at the "engineering." Tools are needed that get existing methods into the hands of testers. These include:

   - static knowledge-based testing tools—for rule-based logics, frame-based logics, and other logics;
   - a "requirements" generator—an automated system that will help a tester generate a requirements document from an examination of the system;
   - benchmarks and other testing tools for shells and inference engines;
   - simple dynamic testing tools (e.g., to keep track of what is going on during the running of the system)—again available for purchase and use;
   - comprehensive tools—such as extensions of the multi-attribute utility analysis tool;
   - integrative tools to tie other tools together.

SPECIFIC ACTIONS

The following are some of the items that could be pursued immediately to advance the state of practice of testing AI.

1. *Develop an anthology on testing AI.* Enough articles have been produced on the subject that a set of reprints on testing AI and expert systems, possibly including a few new articles, could be collected and published as an anthology. This could be done quickly and at low cost, yet provide a useful volume for testers and significant recognition of EPG's efforts and support. At some point, the anthology could be transferred to an electronic medium such as optical disc storage, and a library aid could be developed.

2. *Develop an automated static-tester using di-graph techniques.* There is a pressing need for static software testers for knowledge bases. A promising approach is to develop software that will represent a knowledge base in a directed graph (di-graph). Neural network techniques can then be applied to identify and locate logical faults in the knowledge (including higher-order faults such as multiple link loops).

3. *Develop computerized support for calculating d*, B*.* Chapter 5 presents a method for assessing the predictive accuracy of a knowledge base and defines the parameters d* and B*. Computer programs could be developed for designing tests to collect data for d* and B* (including determining sample sizes). Additionally, such a program could automate statistical calculations involved with d* and B*, automate other aspects of d* and B* calculations (e.g., calculate increase in accuracy possible from the expert system), and automate support to interpret results of statistical tests.

4. *Define a computer program to implement the MAU hierarchy.* As part of this project, we developed prototype software for implementing the MAU hierarchy described in Chapter 3. To be most useful to a tester, this prototype software should be further developed to provide:

   - an automated feature to help a tester build an alternative assessment framework,
   - context-specific advice of the type presented in Chapter 7;
   - human-engineered interfaces
     - built to commercial (or other) standards,
     - more context-specific help,
     - built-in tutorials on related topics;

- ties to other programs, if available or developed (especially d* and B* statistical analyses and di-graph analyzer).

5. *Extend the compendium of lessons learned.* Also as part of this project (Volume 2), we compiled a compendium of lessons learned from testing AI systems in the Army. This is a useful assessment of current practice that can serve to educate testers and to identify areas where additional work is needed. Specific elements of the extended compendium include: surveying new systems developments, extending the coverage of the survey to include additional organizations and systems, and conducting in-depth follow ups on systems reviewed in the existing compendium. The compendium could be produced in both paper and hypertext media.

6. *Extend "testbed" activities.* The methods described in this Handbook have not been applied extensively. The state of practice can advance from applying these test methods in more depth and to more systems. Furthermore, it is possible to build an automated system to collect a historical record of testing including: project and program characterization, testing techniques used, assessment of how well the techniques worked, cost of testing, impact of testing on the schedule, impact of testing on the quality of the system. This history could then form the basis for a knowledge base of testing.

7. *Develop an automated questionnaire generator.* We propose two questionnaires in this Handbook that will help a tester assess the usability of an expert system. Further work in this area might automate the generation of a tailored questionnaire: the tester might input characteristics of the test environment and the system being tested, and a computer program would generate a complete questionnaire. This might then be generalized and linked to other material, such as a regulation or a Mil Std.

8. *Develop methods and tools to validate confidence weights and uncertainty factors.* Many expert systems utilize a formal system to represent confidence or uncertainty (e.g., Bayesian, fuzzy set, or Dempster-Shafer). However, many expert system developers do not understand confidence weights or uncertainty factors (e.g., one survey found many cases of probability assessments outside the range of 0 to 1). Tools could be developed to test whether the knowledge base violates the principles of the confidence or uncertainty handling system that is used.

9. *Interface with software quality factors.* Computer scientists continue to develop software quality factors for standard procedural software. As shown in Chapter 8, there is a relationship between these quality factors and attributes for testing expert systems. This relationship should be explored further with a view toward advancing both approaches for characterizing good software.

CHAPTER SUMMARY

The state-of-the-art in testing expert systems has progressed greatly in the last three or four years from a topic of interest to only a few researchers to a field of application involving many software developers and testers. However, much work is still needed for testing to become a routine part of the expert system development cycle.

This chapter describes a test technology program of acti ities needed to advance both the science and the engineering of expert system testing. All of these activities could be started today, and all will be needed if testing is to play its proper role in expert system applications. This chapter also lists nine immediate actions that could be taken to advance the state of expert system testing generally and the methods described in this Handbook specifically.

REFERENCES

AAAI (1988). *Proceedings of First AAAI Workshop on Validation and Testing of Knowledge-Based Systems.*

Adelman, L. (1982). "Involving Users in the Design of Decision-Analytic Aids: The Principal Factor in Successful Implementation." *Journal of the Operational Research Society, 33,* 333-342.

Adelman, L. (1984). "Real-Time Computer Support for Decision Analysis in a Group Setting: Another Class of Decision Support Systems." *Interfaces, 14,* 75-83.

Adelman, L. (1987). "Supporting Option Generation." *Large Scale Systems, 13,* 83-91.

Adelman, L. (1989). "Measurement Issues in Knowledge Engineering." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-19,* 483-488.

Adelman, L. (1990a). *Integrating Evaluation Methods Into the DSS Development Process.* Information and Decision Technologies.

Adelman, L. (1990b). *Evaluating Decision Support Systems.* Wellesley, MA: QED Information Sciences.

Adelman, L. and M.L. Donnell (1986). "Evaluating Decision Support Systems: A General Framework and Case Study." In S.J. Andriole (Ed.), *Microcomputer Decision Support Systems: Design, Implementation, and Evaluation.* Wellesley, MA: QED Information Sciences.

Adelman, L., M.L. Donnell, R.H. Phelps, and J.F. Patterson (1982). "An Iterative Bayesian Decision Aid: Toward Improving the User-Aid and User-Organization Interfaces." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-12,* 733-742.

Adelman, L. and K. Gates (1983). *Evaluation of the Duplex Army Radio/Radar Targeting Aid (DART)* (Report 83-84). New Hartford, NY: PAR Technology Corporation.

Adelman, L., F.W. Rook, and P.E. Lehner (1985). "User and R&D Specialist Evaluation of Decision Support Systems: Development of a Questionnaire and Empirical Results." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-15,* 334-342.

Adelman, L., P.J. Sticha, and M.L. Donnell (1984). "The Role of Task Properties in Determining the Relative Effectiveness of Multiattribute Weighting Techniques." *Organizational Behavior and Human Performance, 33,* 243-262.

Adelman, L., P.J. Sticha, and M.L. Donnell (1986). "An experimental investigation of the relative effectiveness of two techniques for structuring multi-attributed hierarchies." *Organizational Behavior and Human Decision Processes, 37,* 188-196.

Adelman, L. and J.W. Ulvila (in press). "Evaluating Expert System Technology." In S.J. Andriole and S.M. Halpin (Eds.), *Information Technology For Command and Control*. New York: IEEE Press.

Andriole, S.J. (1989). *Handbook for the Design, Development, Evaluation, and Application of Interactive Military Decision Support Systems*. Princeton, NJ: Petrocelli.

Bahill, A.T., P.N. Harris, and E. Senn (1988). "Lessons Learned Building Expert Systems." *AI Expert*, 3, 36-45.

Bailey, D.E. (1971). *Probability and Statistics: Models for Research*. New York: Wiley & Sons.

Bailey, J.E., and S.W. Pearson (1983). "Development of a Tool for Measuring and Analyzing computer User Satisfaction." *Management Science*, *29*, 530-545.

Baroudi, J.J. and W.J. Orlikowski (1988). "The Problem of Statistical Power in MIS Research." *MIS Quarterly*, 87-106.

Barth, S., D. Sobik, and H. Coyle (1983). *The Users Guide for the Duplex Army Radio/Radar Targeting Aid (DART)*. New Hartford, NY: PAR Technology Corporation.

Beizer, B. (1984). *Software System Testing and Quality Assurance*. New York: Van Nostrand Reinhold.

Bellman, K.L. and E.O. Walter (1988). "Analyzing and Correcting Knowledge-Based Systems Requires Explicit Models." *Proceedings of the AAAI-88 Workshop on Validation and Testing Knowledge-Based Systems*. St. Paul, MN: August 20, 1988.

Bourgeois, L.J. and K.M. Eisenhardt (1988). "Strategic Decision Processes in High Velocity Environments: Four Cases in the Microcomputer Industry." *Management Science*, *34*, 816-835.

Brier, G. (1950). "Verification of Forecasts Expressed in Terms of Probability." *Monthly Weather Review*, *75*, 1-3.

Brown, R.V., A.S. Kahr, and C.R. Peterson (1974). *Decision Analysis for the Manager*. NY: Hot, Rinehart & Winston.

Buede, D.M., and L. Adelman (1987). *Decision Support Systems: Design, Use, and Evaluation*. Coursebook for seminar sponsored by the U.S. Army Logistics Management Center, Fort Lee, VA.

Campbell, D.T. (1984). "Foreword." In R.K. Yin (Ed.), *Case Study Research: Design and Methods*. Beverly Hills: Sage Publications.

Campbell, D.T. and D.W. Fiske (1959). "Convergent and discriminant validation by the multitrait-multimethod matrix." *Psychological Bulletin*, *56*, 81-105.

Campbell, D.T. and J.C. Stanley (1966). *Experimental and Quasi-Experimental Designs for Research*. Chicago, IL: Rand McNally.

Casey, J. (1989). "Picking the Right Expert System Application." *AI Expert*, 4(9), 44-47.

Cats-Baril, W.L. and G.P. Huber (1987). "Decision Support Systems for Ill-Structured Problems: An Empirical Study." *Decision Sciences, 18*, 350-372.

Chandrasekaran, B. (1983). "On Evaluating AI Systems for Medical Diagnosis." *AI Magazine*, 4, 34-37.

Chapnick, P. (1988). "When We Look Back." *AI Expert, 3*(12), 5-6.

Cholawsky, E.M. (1988). "Beating the Prototype Blues." *AI Expert*, 3(12), 42-49.

Cochran, W.G. and G.M. Cox (1957). *Experimental Designs* (2nd edition). New York: Wiley.

Cohen, J. (1965). "Some Statistical Issues in Psychological Research." In B.B. Woleman (Ed.), *Handbook of Clinical Psychology*. NY: McGraw-Hill.

Cohen, J. (1977). *Statistical Power Analysis for the Behavioral Sciences* (revised edition). NY: Academic Press.

Cohen, M.S., and A.N.S. Freeling (1981). *The Impact of Information on Decisions: Command and Control System Evaluation* (Technical Report 81-1). Falls Church, VA: Decision Science Consortium, Inc.

Constantine, M.M. and J.W. Ulvila (1990). "Testing Knowledge-Based Systems: The State of Practice and Suggestions for Improvement." *Expert Systems with Applications, 1*, 237-248.

Cook, T.D. and D.T. Campbell (1979). *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Chicago, IL: Rand McNally.

Culbert, C. and R.T. Savely (1988). "Expert System Verification and Validation." *Proceedings of AAAI-88 Workshop on Validation and Testing Knowledge-Based Systems*. St. Paul, MN. Aug. 20, 1988.

Davis, R. (1989). "Expert Systems: How Far Can They Go." *AI Magazine, 10*, 65-77.

Dawes, R.M. and B. Corrigan (1974). "Linear Models in Decision Making." *Psychological Bulletin, 81*, 95-106.

Delbecq, A.L., A.H. Van de Ven, and D.H. Gustafson (1975). *Group Techniques for Program Planning: A Guide to Nominal Group and Delphi Processes*. Glenview, IL: Scott, Foresman, and Co.

DeMillo, R.A., W.M. McCracken, R.J. Martin, and J.F. Passafiume (1987). *Software Testing and Evaluation*. Menlo Park, CA: The Benjamin/Cummings Publishing Co., Inc.

DOD-STD-1679A: Software Development (Section 5.3, Programming Standards). February 1983.

DOD-STD-2167: Defense System Software Development (Section 30.3, Detailed Requirements section of General Design and Coding Standards). 4 June 1985.

Ebert, R.J. and T.E. Kruse (1978). "Bootstrapping the Security Analyst." *Journal of Applied Psychology, 63*, 110-119.

Edwards, W. (1977). "Use of Multiattribute Utility Measurement For Social Decisions." In D.E. Bell, R.L. Keeney, and H. Raiffa (Eds.), *Conflicting Objectives in Decisions*. New York: Wiley.

Efron, B. (1982). *The Bootstrap, the Jackknife and Other Resampling Plans*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Eils, L.C. and R.S. John (1980). "A Criterion Validation of Multi-Attribute Utility Analysis and of Group Communication Strategy." *Organizational Behavior and Human Performance, 25*, 268-288.

Einhorn, H.J. and R.M. Hogarth (1975). "Unit Weighting Schemes of Decision Making." *Organizational Behavior and Human Performance, 13*, 171-192.

Einhorn, H.J. and W. McCoach (1977). "A Simple Multi-Attribute Procedure for Evaluation." *Behavioral Science, 22*, 270-282.

Eliot, L.B. (1989). "Mass Market Applications: They're Here." *AI Expert*, 4(12), 9-14.

Fairley, R.E. (1985). *Software Engineering Concepts*. New York: McGraw-Hill.

Forsythe, D. and B. Buchanan (1989). "An Empirical Study of Knowledge Elicitation: Some Pitfalls and Suggestions." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-19*, 435-442.

Franklin, W.R., R. Bansal, E. Gilbert, and G. Shroff (1988). "Debugging and Tracing Expert Systems." *International Hawaii Conference on System Sciences*.

Gaschnig, J., P. Klahr, H. Pople, E. Shortliffe, and A. Terry (1983). "Evaluation of Expert Systems: Issues and Case Studies." In F. Hayes-Roth, D.A. Waterman, and D.B. Lenat (Eds.), *Building Expert Systems*. Reading, MA: Addison-Wesley.

Gelperin, D. and B. Hetzel (1988). "The Growth of Software Testing." *Communications of the ACM, 31*, 687-695.

Gilbert, E. (1989). "Static Analysis Tools for Expert Systems." *Proceedings of Test Technology Symposium II*.

Goldberg, L.R. (1970). "Man Versus Model of Man: A Rationale, Plus Some Evidence, for a Method of Improving Clinical Inference." *Psychological Bulletin, 73*, 422-432.

Gould, J.D. and C. Lewis (1985). "Designing for Usability: Key Principles and What Designers Think." *Communications of the ACM, 28*, 300-311.

Green, D., and O. Swets, O. (1966). *Signal Detection Theory and Psychologies*. NY: John Wiley.

Gulliksen, H. (1950). *Theory of Mental Tests*. NY: Wiley and Sons.

Hamlet, R. (1988). "Special Section [illegible] Software Testing." *Communications of the ACM, 31*, 662-667.

Hammond, K.R. (1948). "Subject and Object Sampling: A Note." *Psychological Bulletin, 45*, 530-533.

Hammond, K.R., R.M. Hamm, and J. Grassia (1986). "Generalizing Over Conditions By Combining the Multitrait-Multimethod Matrix and the Representative Design of Experiments." *Psychological Bulletin, 100*, 257-269.

Hammond, K.R., R.M. Hamm, J. Grassia, and T. Pearson (1987). "Direct Comparison of the Relative Efficacy of Intuitive and Analytical Cognition in Expert Judgment. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-17*, 753-770.

Hammond, K.R., T.R. Stewart, B. Brehmer, and D.O. Steinmann (1975). "Social Judgment Theory." M.F. Kaplan and S. Schwartz (Eds.), *Human Judgment and Decision Processes*. New York: Academic Press.

Harmon, P., R. Maus, and W. Morrissey (1988). *Expert Systems Tools and Applications*. New York: John Wiley & Sons.

Harrison, P.R. (1989). "Testing and Evaluation of Knowledge-Based Systems." In J. Liebowitz and D.A. De Salvo (Eds.), *Structuring Expert Systems: Domain, Design, and Development*. Englewood Cliffs, NJ: Yourdon Press.

Hays, W. (1972). *Statistics for the Social Sciences* (2nd Edition). NY: Holt, Rinehart and Winston.

Hayes, P.J. (1981). "The Logic of Frames." B.L. Webber and N.J. Nilsson (Eds.), *Readings in Artificial Intelligence*. Palo Alto, CA: Tioga, 451-458.

Henkaman, D. (1987). "An Empirical Comparison of Three Inference Methods." *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, 158-169.

Hetzel, W. (1984). *The Complete Guide to Software Testing*. Wellesley, MA: QED Information Sciences, Inc.

Hice, G.F., W.S. Turner, and L.F. Cashwell (1978). *System Development Methodology*. New York: [illegible]

Hoffman, P.J. (1960). "The Paramorphic Representation of Human Judgment." *Psychological Bulletin, 57*, 116-131.

Hoffman, P.J., P. Slovic, and L.G. Rorer (1968). "An Analysis-of-Variance Model for the Assessment of Configural Cue Utilization in Clinical Judgment. *Psychological Bulletin, 69*, 338-349.

Hogarth, R.M. (1987). *Judgment and Choice*. NY: Wiley-Interscience.

Huber, G.P. (1980). *Managerial Decision Making*. Glenview IL: Scott, Foresman, & Company.

Huber, G.P. (1986). "The Decision-Making Paradigm of Organizational Design." *Management Science, 32*, 572-589.

Hurst Jr., E.G., D.N. Ness, T.J. Gambino, and T.H. Johnson (1983). "Growing DSS: A Flexible, Evolutionary Approach." In J.L. Bennett (Ed.), *Building Decision Support Systems*. Reading, MA: Addison-Wesley Publishing Company.

IJCAI (1989). *Preliminary Proceedings of IJCAI-89 Workshop on Verification, Validation, and Testing of Knowledge Based Systems*.

JCMPOINST 8020.1: *Independent Software Nuclear Safety Analysis* (Change 2, Appendix F, Section 3.6 (3), Specification and Design Audit and Analysis) (3 March 1984).

Kahneman, D., P. Slovic, and A. Tversky (Eds.) (1982). *Judgment under Uncertainty: Heuristics and Biases*. NY: Cambridge University Press.

Kahneman, D. and A. Tversky (1984). "Choices, Values, and Frames." *American Psychologist, 39*, 341-350.

Kalagnanam, J., and M. Henrion (1988). "A Comparison of Decision Analysis and Expert Rules for Sequential Diagnosis." *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, 205-212.

Kang, Y., and A.T. Bahill (February 1990). "A Tool for Detecting Expert System Errors." *AI Expert*, 5(2), 46-51.

Kaplan, B. and D. Duchon (1988). "Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study." *MIS Quarterly, 12*, 571-586.

Keeney, R.L. and H. Raiffa (1976). *Decisions with Multiple Objectives*. New York: Wiley.

Keim, R.T. and R. Janaro (1982). "Cost/Benefit Analysis of MIS." *Journal of Systems Management*, September 1982, 20-25.

Kelly, C.W. (1979). *Program Completion Report: Advanced Decision Technology Program (1972-1979)* (TR 79-3-93). McLean, VA: Decisions and Designs, Inc.

Keyes, J. (1989). "The Citibank Pension Expert." *AI Expert*, 4(6), 61-65.

Kirk, D.B. and A.E. Murray (1988). *Verification and Validation of Expert Systems for Nuclear Power Applications*. McLean, VA: Science Applications International Corporation.

Klein, G.A. and C. Brezovic (1988). "Evaluation of Expert Systems." In S.J. Andriole and G.W. Hopple (Eds.), *Defense Applications of AI*. Lexington Books.

Kraemer, H.C. and S. Thiemann (1987). *How Many Subjects?: Statistical Power Analysis in Research*. Beverly Hills, CA: Sage Publications.

Lay, P.M.W. (1985). "Beware of the Cost/Benefit Model for IS Project Evaluation." *Journal of Systems Management*, June 1985, 30-35.

Leddo, J.M., and M.S. Cohen (1987). "A Cognitive Science Approach to Elicitation of Expert Knowledge." *Proceedings of the 19[illegible] Directors of Laboratories Conference*. McLean, VA: [illegible] International Corporation.

Lee, A.S. (1989). "A Scientific Methodology for MIS Case Studies." *MIS Quarterly, 13*, 33-50.

Lehner, P.E. (1989). "Toward an Empirical Approach to Evaluating the Knowledge Base of an Expert System." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-19*, 658-662.

Lehner, P.E. and L. Adelman (in press). "Behavioral Decision Theory and Its Implications for Knowledge Engineering." *Knowledge Engineering Review*.

Lehner, P.E., T.M. Mullin, and M.S. Cohen (1989). "When Should a Decision Maker Ignore the Advice of a Decision Aid." *Proceedings of the 1989 Workshop on Uncertainty in Artificial Intelligence*.

Lehner, P.E. and J.W. Ulvila (1989). *A Note on the Application of Classical Statistics to Evaluating the Knowledge Base of an Expert System*. Reston, VA: Decision Science Consortium, Inc.

Lehner, P.E., and D.A. Zirk (February 1987). "Cognitive Factors in User/Expert System Interaction. *Human Factors, 29*(1), 97-109.

Levi, K. (1985). "A Signal Detection Framework for the Evaluation of Probabilistic Forecasts." *Organizational Behavior and Human Performance, 36*, 143-166.

Levi, K. (1989). "Expert Systems Should be More Accurate than Human Experts: Evaluation Procedures from Human Judgment and Decisionmaking." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-19*, 647-657.

Libby, R. and B.L. Lewis (1977). "Human Information Processing Research in Accounting." *Accounting, Organizations, and Society, 21*, 245-268.

[illegible]owitz, [illegible] "Useful Approaches for Evaluating Expert Systems." *Expert Systems*, [illegible] 86-95.

Likert, R. (1932). "A Technique for the Measurement of Attitudes. *Arch. Psychol., 140*.

Marcot, B. (1987). "Testing [illegible] Knowledge Base." *AI Expert, 2*, 42-47.

Markus, M.L. (1983) "Power, Politics and MIS Implementation." *Communications of the ACM, 26*, 430-444.

Markus, M.L. (1984). *Systems in Organizations: Bugs and Features*. Marshfield, MA. Pitman Publishing, Inc.

Mazen, A., L. Graf, C. Kellogg, and M. Hermmasi (1987). "Statistical Power in Contemporary Management Research." *Academy of Management Journal, 30*, 369-380.

McCain, L.J. and R. McCleary (1979). "The Statistical Analysis of the Simple Interrupted Time-Series Quasi-Experiment." In T.D. Cook and D.T. Campbell (Eds.), *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Chicago, IL: Rand McNally.

McCall, J. and M. Matsumoto (1980). *Software Quality Metrics Enhancement: Volumes I-II* (RADC-TR-80-109).

Medlin, S.M., and L. Adelman (1989). "Automated cost-benefit analysis: A powerful decision support tool for HRD managers." *Proceedings of the Twelfth National Conference on Teaching Public Administration*, Charlottesville, VA: March 14-16, 377-392.

MIL-STD-1679: Weapon System Software Development (Section 5.3, Programming Standards). 1 Dec. 1978.

Mortimer, H. (1988). *The Logic of Induction* (English edition). Chicester, England: Ellis Horwood Limited.

Nazareth, D.L. (1988). *An Analysis of Techniques for Verification of Logical Correctness in Rule-Based Systems* (Ph.D. Dissertation). Case Western Reserve University.

Nazareth, D.L. (1989). "Issues in the Verification of Knowledge in Rule Based Systems." *International Journal of Man-Machine Studies, 30*, 255-271.

Newquist, H.P. III (1988). "Tales from the Hearth of AI." *AI Expert, 3*(12), 61-63.

Nguyen, T.A., W.A. Perkins, T.J. Laffey, and D. Pecora (1987). "Knowledge Base Verification." *AI Magazine, 8*, 69-75.

Nilsson, N. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann.

Noble, D. (1989). "Schema-Based Knowledge Elicitation for Planning and Situation Assessment Aids." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-19*, 473-482.

O'Connor, M.F. (1989). "Planning for Integrated System Evaluation: An Application to SDI." In S.E. Johnson and A.H. Levis (Eds.), *Science of Command and Control: Coping With Complexity* (Part II). Fairfax, VA: AFCEA International Press.

O'Connor, M.F. and W. Edwards (1976). *On Using Scenarios in the Evaluation of Complex Alternatives* (DDI/DT/TR-76-17). McLean, VA: Decisions and Designs, Inc.

O'Keefe, R.M., O. Balci, and E.P. Smith (1987). "Validating Expert System Performance." *IEEE Expert, 2*, 81-90.

O'Keefe, R.M. and O'Leary, D.E. (1990). *The Verification and Validation of Expert Systems* Authors' mimeo.

Pitz, G.F. and J. McKillip (1984). *Decision Analysis for Program Managers*. Beverly Hills, CA: Sage Publications.

Press, L. (1989). "Expert System Benchmarks." *IEEE Expert*, 4, 37-44.

Pressman, R.S. (1982). *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill.

Reichardt, C.S. (1979). "The Statistical Analysis of Data from Nonequivalent Group Designs." In T.D. Cook and D.T. Campbell (Eds.). *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Chicago, IL: Rand McNally.

Riedel, S.L. and G.F. Pitz (1986). "Utilization-Oriented Evaluation of Decision Support Systems." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-16*, 980-996.

Rockmore, A.J., L. Hemphill, R.A. Riemenschneider, M.L. Donnell, and K. Gates (1982). *Decision Aids for Target Aggregation: Technology Review and Decision Aid Selection* (PAR Report #82-32). New Hartford, New York: PAR Technology Corporation.

Rook, F.W. and J.W. Croghan (1989). "The Formulation of Knowledge Acquisition Methods: A Systems Engineering Conceptual Framework." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-19*, 586-597.

Rushby, J. (1988). *Quality Measures and Assurance for AI Software* (NASA Contractor Report 4187). Washington, DC: National Aeronautics and Space Administration (Code NTT-4).

Saaty, T.L. (1980). *The Analytic Hierarchy Process*. New York: McGraw-Hill.

Sage, A.P. (1986). "An Overview of Contemporary Issues in the Design and Development of Microcomputer Decision Support Systems." In S.J. Andriole (Ed.), *Microcomputer Decision Support Systems: Design, Implementation, and Evaluation*. Wellesley, MA: QED Information Sciences.

[illegible], A.S. and C.C. White III (1980). *Evaluation of Two DDI Decision Aids Developed for [illegible]* (Document No. 3375?-Wil4-RU-00). Falls Church, VA: [illegible] Defense and Space Systems Group.

Schein, E.H. (1970). *Organizational Psychology*. Englewood Cliffs, NJ: Prentice-Hall, 1970.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton, NJ: Princeton University Press.

Shank, R.C. and R.P. Abelson (1977). *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Sharda, R., S.H. Barr, and J.C. McDonnell (1988). "Decision Support System Effectiveness: A Review and an Empirical Test." *Management Science*, 34, 139-159.

Sheppard, J. (November 1989). "An Approach to Verifying Expert System Rule Bases." *Proceedings of the 1989 International Conference on Systems, Man, and Cybernetics*.

Shycon, H.N. (1977). "All Around the Model—Perspective on MS Applications." *Interfaces*, 7, 40-43.

Simon, H.A. (1960). *The New Science of Management Decisions*. New York: Harper & Row.

Sizemore, N.L. (1990). "Test Technologies for Knowledge-Based Systems: A Summary." *Proceedings of Test Technology Symposium III*. Aberdeen, MD: U.S. Army Test and Evaluation Command.

Slagle, J.R. and M.R. Wick (1988). "A Method for Evaluation Candidate Expert System Applications." *AI Magazine*, *9*, 44-53.

Smith, D.L. (1988). "Implementing Real World Expert Systems." *AI Expert*, 3(12), 51-57.

Software Quality Engineering Handbook (1984). Draft Technical Bulletin TB-18-102-2.

Spetzler, C.S. and C.A.S. Stael von Holstein, (1975). "Probability Encoding in Decision Analysis." *Management Science*, *22*, 340-358.

Stachowitz, R.A., C.L. Chang, and J.B. Combs (1988). "Research on Validation of Knowledge-Based Systems." *Proceedings of the AAAI-88 Workshop on Validation and Testing Knowledge-Based Systems*. St. Paul, MN: Aug. 20, 1988.

Stachowitz, R.A., and J.B. Combs (1987). "Validation of Expert Systems." *Proceedings, Hawaii International Conference on System Sciences*, Kong, HI.

Stewart, T.R., W.R. Moninger, J. Grassia, R.H. Brady, and F.H. Merrem (1988). *Analysis of Expert Judgment and Skill in a Hail Forecasting Experiment*. Boulder, CO: Center for Research on Judgment and Policy at the University of Colorado.

Tong, R.M., N.D. Newman, G. Berg-Cross, and F. Rook, (1987). *Performance Evaluation of Artificial Intelligence Systems*. Mountain View, CA: Advanced Decision Systems.

Ulvila, J.W. and Chinnis, J.O., Jr. (in press). Decision analysis for R&D resource management. In D.F. Kocaoglu (Ed.), *Management of R&D and engineering*.

Ulvila, J.W., P.E. Lehner, T.A. Bresnick, J.O. Chinnis, Jr., and J.D.E. Gumula (1987). *Testing and Evaluating C^3I Systems That Employ Artificial Intelligence*. Reston, VA: Decision Science Consortium, Inc.

von Winterfeldt, D. and W. Edwards (1986). *Decision Analysis and Behavioral Research*. NY: Cambridge University Press.

Watson, S. and D.M. Buede (1987). *Decision Synthesis: The Principles and Practice of Decision Analysis*. Cambridge, England: Cambridge University Press.

Weiss, J.J. and G.W. Zwahlen (1982). "The Structured Decision Conference. A Case Study." *Hospital and Health Services Administration*, *27*, 90-105.

Weitzel, J.R. and L. Kerschberg (1989). "A System Development Methodology for Knowledge-Based Systems." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-19*, 598-605.

Winkler, R. (1972). *Introduction to Bayesian Inference and Decision.* NY: Holt, Rinehart and Winston.

Wohl, J.G. (1981). "Force Management Decision Requirements for Air Force Tactical Command and Control." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-11*, 618-639.

Wolfgram, D.D., T.J. Dear, and C.S. Galbraith (1987). *Expert Systems for the Technical Professional.* New York: Wiley.

Yin, R.K. (1984). *Case Study Research: Design and Methods.* Beverly Hills: Sage Publications.

Yu, V.L., L.M. Fagan, S.M. Wraith, W.J. Clancey, A.C. Scott, J.F. Hanigan, R.L. Blum, B.G. Buchanan, and S.N. Cohen (1979). "Antimicrobial Selection by a Computer: A Blinded Evaluation by Infectious Disease Experts." *Journal of the American Medical Association, 242*, 1279-1282.

Zimmerman, H.J. and P. Zysno (1980). "Latent Connectives in Human Decision Making." *Fuzzy Sets and Systems, 4*, 37-51.

APPENDIX: QUESTIONNAIRES

This appendix contains generic questionnaires that can be used by a tester to elicit judgments from subjects on the performance and usability of an expert system. These questionnaires have been designed using the guidelines described in Chapter 3, and are directed toward assessing attributes in the MAU framework for testing and evaluating expert systems that is described in Chapter 3, using the 100-point scale that is also described in Chapter 3.

Two types of questionnaire are included. The first uses a Likert-type scale for responses to agreement or disagreement with statements. At least two questions, which appear in different places and are phrased differently, are included for each attribute. These responses must be converted to utility scores by the tester. We expect that a subject should need 15 to 30 minutes to complete this questionnaire. Figure A-1 shows the relationship between the questions and the hierarchy of attributes. The second questionnaire asks the subject to make judgments on a utility scale for fifteen attributes. This scale is described, and instructions are also given on how to fill out the questionnaire. Subjects will probably need more time to complete this questionnaire. If resources permit, we recommend the use of both questionnaires.

Both questionnaires should be customized for each expert system being tested. This can be done by replacing the following generic phrases with specific ones where or they appear. The generic phrases are underlined throughout the questionnaires to aid in this change.

Generic Phrase	Replace With
"the expert system"	the name of the expert system being tested
"expert system's task"	a description of the function that the expert system performs
"the organization"	the name of the organization that will use the expert system

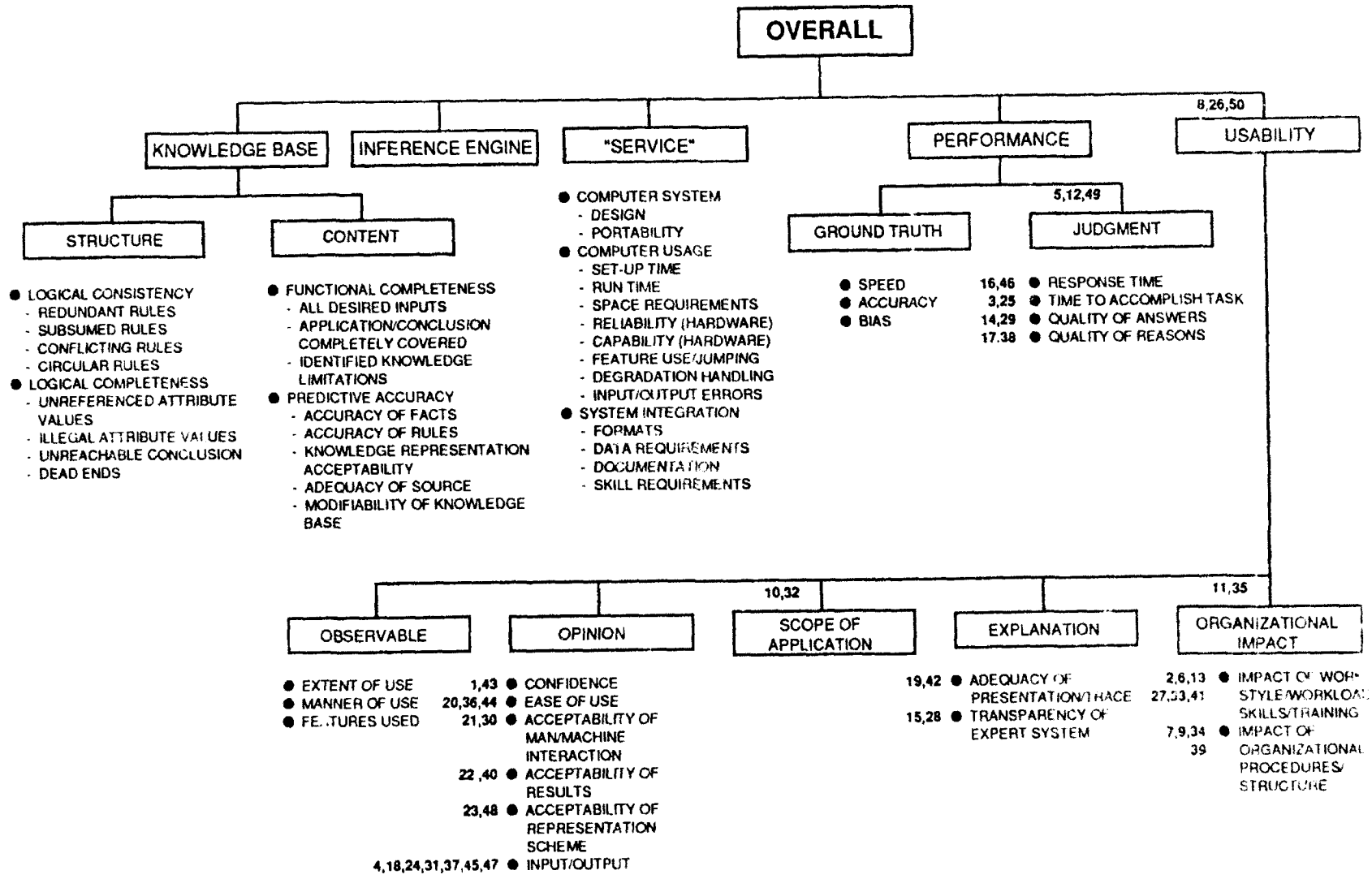


Figure A-1: Relationship between the Questionnaire and the MAU Framework for Testing and Evaluating Expert Systems

Name: _____ Date: _____

Time Started this Questionnaire: _____

Time Finished this Questionnaire: _____

Sequence: _____

QUESTIONNAIRE

The purpose of this questionnaire is to obtain your perceptions concerning the value of the expert system in performing the expert system's task for which the system has thus far been developed. A number of statements are made and each is followed by an 11-point scale on which you are to indicate the extent to which you agree or disagree with the statement. Please simply mark an X at the appropriate point on the scale. If you cannot answer a question, put an asterisk(*) in the right-hand margin beside the question. If you would like to comment about your answer, please do so in the space provided to the right of the question. Comments are helpful, but they are not required.

There are 50 questions in the questionnaire. We have to ask you so many questions in order to get a complete and accurate picture of your perceptions concerning the strengths, weaknesses, and potential value of the expert system. You should be able to complete the questionnaire within 15 to 30 minutes, but please take as much time as you need to carefully and accurately respond to each question.

1. I have a lot of confidence in the expert system.

Comments

Very Strongly Disagree					Neither Disagree Nor Agree					Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

2. Using the expert system will significantly decrease the workload required by personnel performing the expert system's task.

Comments

Very Strongly Disagree					Neither Disagree Nor Agree					Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

3. The expert system's task can be performed faster using the expert system.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

4. The expert system's input displays are acceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

5. In general, I am pleased with the expert system's overall level of performance.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

6. On the average, the expert system nicely matches the background and skills of the organization's personnel performing the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

7. The expert system will disrupt communication or the flow of data among personnel performing the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

8. In general, the expert system's overall usability is good.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

9. The expert system would have an unacceptable impact on organizational structure and procedures.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

10. The expert system ignores situations, data, or predictions that are important when performing the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

11. The expert system matches the work style, workload, skills, and training of the people who will use it.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

12. I think the expert system's overall level of performance is unacceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

13. It would be difficult to train personnel to use the expert system effectively.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

14. I could produce higher quality results working with the expert system than without it.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

15. I do not fully understand how the expert system works.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

16. In general, the time between my input and the system's response is fast and quite acceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

17. The expert system's built-in expertise is worthless for the organization's personnel performing the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

18. The expert system's input and output displays are not very good.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

19. The presentation of the expert system's reasoning process is adequate.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

20. I find the expert system difficult to use.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

21. The expert system does not split the tasks between the machine and the user appropriately.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

22. I find the expert system's results acceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

23. The expert system uses an objectionable scheme for representing and accessing knowledge.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

24. The expert system's output displays are acceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

25. The use of the expert system will slow down the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

26. Regardless of whether the expert system performs well or not, I do not like using it.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

27. The expert system does *not* match the knowledge of personnel performing the expert system's task.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

28. The problem-solving logic used by the expert system is transparent to the user.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

29. The expert system fails to give high-quality answers.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

30. The expert system has been designed so that the operator and the system are doing the tasks for which they are best suited.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

31. The expert system's interface with the user is very good.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

32. The expert system has a broad enough scope of application to be useful in performing the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

33. The expert system will increase the amount of work required to perform the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

34. The expert system would improve communication.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

35. The expert system will adversely impact the work style, workload, skills, or training of those personnel it was designed to support.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

36. Once I have completed training and gained familiarity with the system, I can easily and effectively use it.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

37. The mechanics of using the expert system causes problems in data entry.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

38. I think that the expert system gives high-quality justifications for its answers.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

39. The expert system is well-matched to the organizational structure and characteristics of the organization.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree				Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

40. After using the expert system, I find its results to be unacceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree				Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

41. The use of the expert system would reduce the amount of time required to train new staff members to perform the expert system's task.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree				Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

42. The expert system does *not* help the operator understand how different characteristics of the situation might result in different outputs.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree				Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

43. I am apprehensive in taking actions based on the expert system.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

44. Once I have completed training and gained familiarity with the expert system, I can easily and effectively use it without consulting the documentation or members of the development team.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

45. The words and phrases used in the expert system are appropriate.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

46. I have to wait too long for the expert system to respond to my inputs.

Comments

Very Strongly Disagree			Neither Disagree Nor Agree							Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

47. The expert system's graphic displays and tables, and hardcopy capabilities for printing displays, are unacceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

48. The scheme used by the expert system to represent knowledge is acceptable.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

49. When I consider, in total, (1) the expert system's response time, (2) the amount of time to accomplish the task with the expert system, (3) the quality of the expert system's answers, and (4) the quality of its reasons, I think the expert system is generally performing well.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

50. The expert system scores well on overall usability, which includes its user interface, its match with users' background, and its match with the organization's procedures and operations.

Comments

Very Strongly Disagree				Neither Disagree Nor Agree						Very Strongly Agree
0	1	2	3	4	5	6	7	8	9	10

Name: _____ Date: _____

Time Started this Questionnaire: _____

Time Finished this Questionnaire: _____

Sequence: _____

INSTRUCTIONS FOR SCORING THE EXPERT SYSTEM ON THE EVALUATION CRITERIA

We need your assistance in evaluating (or "scoring") the expert system on the following criteria, which are bold and in capital letters, that might be considered judgmental in nature:

1. The adequacy of the system's **Response Time Performance**, in terms of the amount of time the expert system takes to respond to the operator's inputs and provide outputs.

2. The system's **Performance** with respect to the overall **Time to Accomplish** the expert system's **Task**.

3. **Quality** of the expert system's **Answers**.

4. **Quality of Reasons** given for answers.

5. **Your Confidence** in taking actions based on using the expert system.

6. **How Easy** it is **To Use** the expert system after you had been completely trained and familiar with the system.

7. **Acceptability of the Person-Machine Interaction Process**, in terms of the different tasks assigned to the operator and to the system.

8. **Acceptability of** the expert system's task **Results**.

9. **Acceptability of** the system's **Scheme for Representing Knowledge**.

10. **Acceptability of** the expert system's **Scope of Application** for the expert system's task.

11. **Acceptability of** the system's **Presentation (or Trace) for Explaining the Reasons** for its answers.

12. Transparency of the expert system.

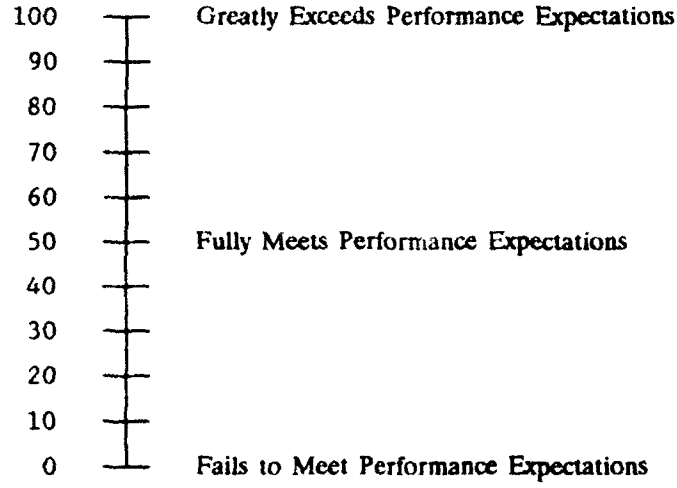
13. **Acceptability of** the system's **Impact on the Work Style, Workload, Skills, and Training** of personnel performing the expert system's task.

14. **Acceptability of** the system's **Impact on Organizational Procedures and Structures** of the organization.

15. **Acceptability of** the expert system's **Input-Output Capabilities**--that is, all the system's displays except those tracing the reasoning process.

Feel free to refer back to these definitions when completing the questionnaire.

We would like you to use the following "0 to 100" scale to score the expert system on each evaluation criterion:



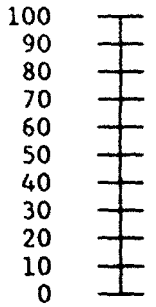
The "50" point means that the expert system fully meets your performance expectations for the system on the evaluation criterion being considered. The "0" means the system fails to meet your performance expectations. The "100" means the system not only fully meets your performance expectations, but greatly exceeds them.

More generally, scores below "50" mean that, in your judgment, the expert system is in some fashion deficient on the criterion; scores above "50" mean that you think the expert system is providing added value on the criterion. The scale permits you to numerically score the level of deficiency or the level of added value. For example, let's consider evaluating the system on **Response Time**. If the expert system meets your performance expectations for an acceptable waiting period between your inputs and the system's response to them, you would give it a score of "50." Let's assume that you consider the expert system's response time deficient (i.e., less than "50"), but not a complete failure (i.e., greater than "0"). Then, the question is, "What is its numerical level of deficiency between 0 and 50?" If the deficiency is very, very minor in your mind, then the score would be close to 50 (e.g., greater than or equal to 45, but less than 50). On the other hand, if the deficiency is very, very great, but still not "0," then the score would be close to 0 (e.g., less than or equal to 5, but greater than 0). If you think the level of deficiency is about halfway between meeting the expectation and failing it, you would give the system a score of 25; if it is a quarter-of-the-way, you would score it 12.5, and so forth. In short, you can use the bottom-half of the scale to numerically specify the expert system's level of deficiency on the evaluation criterion. In addition, of course, we need to know the reason(s) for your score; consequently, we will provide you with space to tell us.

In a similar fashion, you can use the scale between "50" and "100" to numerically specify the level of "added value" performance on the criterion. For example, if you think the expert system barely exceeds your performance expectations for **Response** Time, then it would receive a score slightly above "50." If it considerably exceeds your performance expectations but is not a "100," the expert system might receive an "85," "90," "95," etc. If the degree of added value benefit provided by the expert system is about halfway between meeting your performance expectations and greatly exceeding it, then you would score it "75." If the added-value benefit is a quarter-of-the-way, you would give it a score of "62.5;" if it is three-quarters, you would score it "87.5," and so forth. Again, it is important to tell us why the expert system is providing added-value on the criterion; that is, give us the reasons for your score.

The pages that follow provide scoring sheets for evaluating the expert system on each criterion. Please think carefully about each score, and the reasons for it. One way to help you do this is for you to first carefully think about your performance expectations for the system on the criterion. What level of performance do you consider acceptable (i.e., a score of "50")? We have given you room to write your performance expectations on the scoring sheet. We have also given you an opportunity to indicate (a) whether you have previously expressed performance expectations for the criterion, and (b) whether you have heard anyone else express performance expectations. Then, provide a numerical score, and the reasons for it, in the space provided. If you *can* not (or *do* not want to) score the expert system on a particular criterion, please write "no response" in the space provided.

1. RESPONSE TIME PERFORMANCE



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no.")

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

  _____

  _____

  _____

  _____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

  _____

  _____

  _____

  _____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

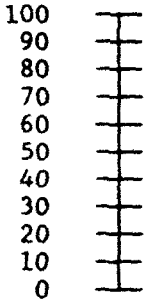
  _____

  _____

  _____

  _____

2. PERFORMANCE REGARDING TIME TO ACCOMPLISH TASK



Greatly Exceeds Performance Expectations

Fully Meets Performance Expectations

Fails to Meet Performance Expectations

- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE _____

- REASONS FOR SCORE:

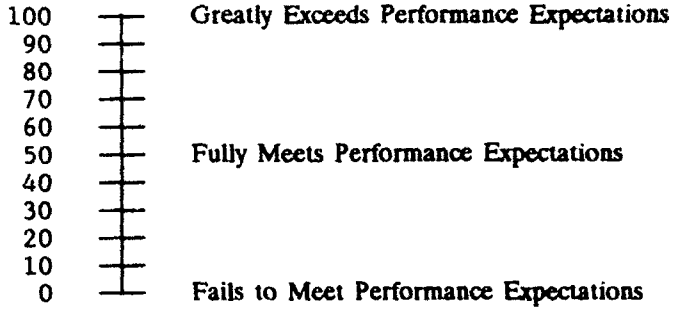
_____

_____

_____

_____

3. QUALITY OF ANSWERS



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

  _____
  _____
  _____
  _____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

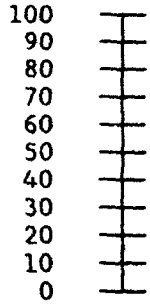
  _____
  _____
  _____
  _____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

  _____
  _____
  _____
  _____

4. QUALITY OF REASONS



Greatly Exceeds Performance Expectations

Fully Meets Performance Expectations

Fails to Meet Performance Expectations

- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

  _____

  _____

  _____

  _____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

  _____

  _____

  _____

  _____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

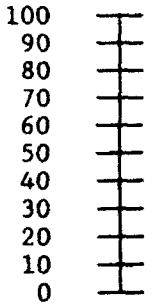
  _____

  _____

  _____

  _____

5. YOUR CONFIDENCE



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

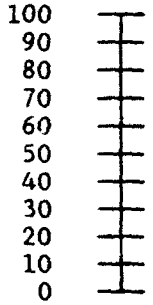
_____

_____

_____

_____

6. EASE OF USE



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

  _____

  _____

  _____

  _____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

  _____

  _____

  _____

  _____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

  _____

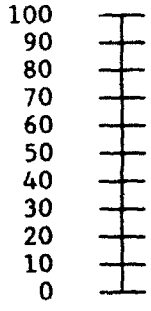
  _____

  _____

  _____

7. ACCEPTABILITY OF THE PERSON-MACHINE INTERACTION PROCESS

(In terms of the different tasks assigned to the operator and to the system)



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

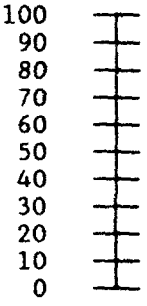
_____

_____

_____

_____

8. ACCEPTABILITY OF RESULTS



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

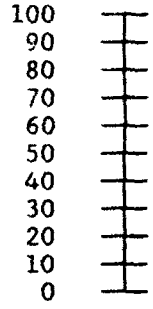
_____

_____

_____

_____

9. ACCEPTABILITY OF SCHEME FOR REPRESENTING KNOWLEDGE



Greatly Exceeds Performance Expectations

Fully Meets Performance Expectations

Fails to Meet Performance Expectations

- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

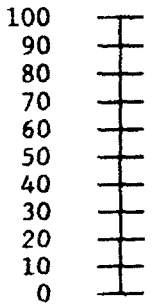
_____

_____

_____

_____

10. SCOPE OF APPLICATION



Greatly Exceeds Performance Expectations

Fully Meets Performance Expectations

Fails to Meet Performance Expectations

- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

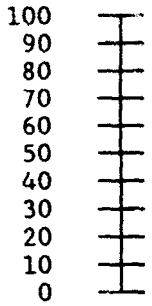
_____

_____

_____

_____

11. ADEQUACY OF PRESENTATION/TRACE FOR EXPLAINING REASONS



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

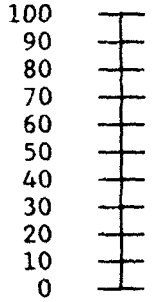
_____

_____

_____

_____

12. TRANSPARENCY OF THE EXPERT SYSTEM



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

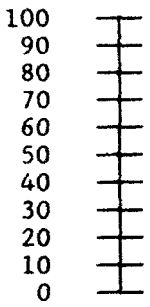
_____

_____

_____

_____

13. ACCEPTABILITY OF IMPACT ON OPERATOR'S WORK STYLE, WORKLOAD, SKILLS, AND TRAINING



Greatly Exceeds Performance Expectations

Fully Meets Performance Expectations

Fails to Meet Performance Expectations

- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

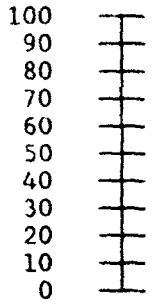
_____

_____

_____

_____

14. ACCEPTABILITY OF IMPACT ON ORGANIZATIONAL PROCEDURES AND STRUCTURES



Greatly Exceeds Performance Expectations

Fully Meets Performance Expectations

Fails to Meet Performance Expectations

- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

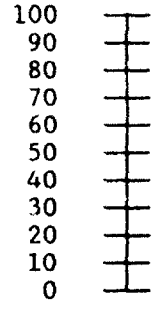
_____

_____

_____

_____

15. ACCEPTABILITY OF INPUT-OUTPUT CAPABILITIES



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

_____

_____

_____

_____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

_____

_____

_____

_____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

_____

_____

_____

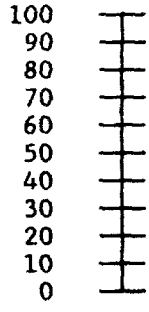
_____

Now we would like you to evaluate the expert system's **Overall Performance**. When doing so, please consider the expert system's performance on the following four criteria *taken together*.

- Response Time Performance
- Time to Accomplish the Task
- Quality of the Expert System's Answers
- Quality of the Expert System's Reasons.

Now please turn the page and score the expert system on its **Overall Performance**. Feel free to look back at your previous scores on the four criteria listed above before scoring the expert system on **Overall Performance**.

16. OVERALL PERFORMANCE



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

  _____

  _____

  _____

  _____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

  _____

  _____

  _____

  _____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

  _____

  _____

  _____

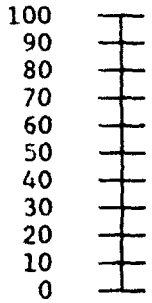
  _____

Now we would like you to evaluate the expert system on its **Overall Usability**. When doing so, please consider the expert system's performance on the following eleven criteria *taken together*:

- Your Confidence in the Expert System
- The Expert System's Ease of Use
- Acceptability of its Person-Machine Interaction Process
- Acceptability of its Results
- Acceptability of the Expert System's Scheme for Representing Knowledge
- The Expert System's Scope of Application
- Adequacy of Presentation/Trace for Explaining Reasons
- Transparency of the Expert System
- Acceptability of its Impact on Operator's Work Style, Workload, Skills, and Training
- Acceptability of its Impact on Organizational Procedures and Structures
- Acceptability of the Expert System's Input-Output Capabilities.

Now please turn the page and score the expert system on its **Overall Usability**. Feel free to look back at your previous scores on the eleven criteria listed above before scoring the expert system on **Overall Usability**.

17. OVERALL USABILITY



- Have you previously expressed performance expectations for this criterion? (Circle "yes" or "no".)

  Yes No

- Have you previously heard anyone else express performance expectations? If yes, please comment.

  Yes No

  _____

  _____

  _____

  _____

- WHAT ARE YOUR PERFORMANCE EXPECTATIONS?

  _____

  _____

  _____

  _____

- NUMERICAL SCORE: _____

- REASONS FOR SCORE:

  _____

  _____

  _____

  _____