AD-A185 800

(2)

Evaluating Expert System Tools

A Framework and Methodology

Jeff Rothenberg, Jody Paul, Iris Kameny,
James R. Kipps, Marcy Swenson

RAND

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER R-3542-DARPA	2. GOVT ACCESSION NO. ADA185800	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Evaluating Expert System Tools: A Framework and Methodology		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. Rothenberg, J. Paul, I. M. Kameny, J. R. Kipps, M. Swenson		8. CONTRACT OR GRANT NUMBER(s) MDA903-85-C-0030
9. PERFORMING ORGANIZATION NAME AND ADDRESS The RAND Corporation 1700 Main Street Santa Monica, CA 90406		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Department of Defense Arlington, VA 22209		12. REPORT DATE July 1987 13. NUMBER OF PAGES 55
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release - Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No Restrictions

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence
Computer Programming
Computer Programs
Test and Evaluation

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

see reverse side

DD FORM 1 JAN 73 1473

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This report summarizes the results of a study undertaken to develop criteria for evaluating and selecting tools used to build expert systems. The authors used an evaluation framework composed of five elements: (1) application characteristics, which describe the problem and the project to be undertaken; (2) tool capabilities, the capabilities that the tools support; (3) metrics, the quantitative and qualitative measures of merit for expert system tools; (4) assessment techniques, specific ways of applying metrics to tools; and (5) contexts, which describe the ways in which the evaluation criteria depend on the development phases targeted by a project. Many of the study's conclusions relate to software engineering aspects of the expert system endeavor. Robustness, reliability, portability, integrability, database access, concurrent access, performance, and user interface all appear to be increasingly important requirements for tools, as well as eventual requirements for the expert systems that will be produced with those tools. In addition, the expert system paradigm seems to have had a significant and beneficial effect on software engineering itself.

R-3542-DARPA

Evaluating Expert System Tools

A Framework and Methodology

Jeff Rothenberg, Jody Paul, Iris Kameny,
James R. Kipps, Marcy Swenson

July 1987

Prepared for the
Defense Advanced Research Projects Agency

RAND

PREFACE

This report summarizes the results of a study conducted for the Information Science and Technology Office of the Defense Advanced Research Projects Agency (DARPA), under RAND's National Defense Research Institute (NDRI). The NDRI is a Federally Funded Research and Development Center sponsored by the Office of the Secretary of Defense. The study was undertaken to develop criteria for evaluating and selecting tools used to build expert systems. The report should be of interest primarily to decisionmakers concerned with choosing such tools, i.e., managers of expert system development projects and developers of expert systems. It should also be useful for developers of expert system tools and artificial intelligence (AI) researchers investigating new expert system techniques.

This work draws heavily on the experience of expert system tool developers and users. The authors enhanced their own background in the field by studying and using a number of major tools, and by hosting two workshops: one for tool developers (representing seven commercial vendors) and one for tool users (representing over thirty expert system development projects). The workshop discussions validated and refined the authors' ideas and provided both objective and anecdotal evidence about the state of current expert system tools and expert system research and development in general. The workshops are described in companion RAND Note N-2603-DARPA, *Evaluating Expert System Tools: A Framework and Methodology—Workshops*, by J. Rothenberg, J. Paul, I. Kameny, J. Kipps, and M. Swenson. Related material in the Note is cross-referenced throughout this report.

SUMMARY

Expert systems represent a new approach to solving problems with computers, using programs that explicitly embody human knowledge and expertise from a given problem domain. The expert system paradigm emphasizes the rapid generation of prototype systems whose behavior can be understood and refined by domain experts. Because the expert system approach differs from traditional software engineering, it has spawned a new class of tools that can provide considerable leverage in building expert systems. One of the first steps an expert system developer usually takes is therefore to survey the available tools and decide which, if any, is most appropriate to the task at hand. However, this evaluation is a complex task; its cost and the attendant risk of performing it ineffectively motivate the development of a rational, reliable strategy for evaluating expert system tools. This report develops a framework of criteria for performing such evaluations, along with a methodology for tailoring and applying this framework to given projects and problems.

Our evaluation framework consists of five dimensions:

- Application characteristics
- Tool capabilities
- Metrics
- Assessment techniques
- Contexts

The *application characteristics* describe the problem and the project to be undertaken. Problem characteristics include the nature of the problem domain, the chosen problem within that domain, the available sources of expertise, and the nature of the target environment for the proposed system. Project characteristics include the scope of the project, the development environment in which it will be undertaken, and the nature of the proposed development team.

Tool capabilities are the capabilities the tools support. We focus on the capabilities rather than the features of the tools, because capabilities provide a more stable basis for evaluation. It must be noted, however, that even capabilities are volatile, due to the rapid evolution of expert system technology.

Metrics are quantitative and qualitative measures of merit for expert system tools. We describe six aggregated metrics: cost, flexibility, extensibility, clarity, efficiency, and vendor support.

Assessment techniques are specific ways of applying metrics to tools. We identify a number of candidate techniques, including direct comparisons, benchmarks, interviews, questionnaires, personal advice, libraries of case studies and development efforts, and knowledge-based systems to help evaluate tools.

Contexts describe the ways in which the evaluation criteria depend on the development phases targeted by a project, i.e., whether a tool will be used for conceptualizing, prototyping, developing, delivering, or maintaining an expert system.

Users of expert system building tools generally agree that the tools are of significant value and that they provide a great advantage over building expert systems directly in a programming language (such as LISP). Most users feel that current tools are well-designed, reasonably supported, and sufficiently powerful to justify their cost. The most frequently cited shortcomings of tools are lack of speed, lack of explicit control over inferencing capabilities, and the tendency of vendors to release new versions that have not been rigorously debugged.

Building an expert system is essentially a software engineering endeavor. This is reflected in the importance of integrability, reliability, and modifiability in the design of expert systems. The expert system paradigm is having a salutary effect on software engineering by emphasizing the importance of the explicit, declarative representation of knowledge, as opposed to an implicit, procedural approach, and by advocating a development strategy based on iterative prototyping and refinement, as opposed to sequential specification, design, and implementation.

ACKNOWLEDGMENTS

The authors would like to thank the many researchers, tool developers, and tool users whose ideas and insights contributed to this project. In seeking to collect the experiences of this large group, we encountered a gratifying spirit of inquiry and concern for the benefit of the field as a whole. The workshops we held at RAND brought together representatives from competing vendor and user organizations who were able to join intellectual forces in a highly cooperative and productive display of professionalism. In addition to those who attended our workshops, many others whom we could not accommodate nevertheless often provided considerable insight and support for our endeavor and were kind enough to forgive us for our logistic constraints that kept them from participating more fully. We greatly appreciate their help and understanding. The organization of the workshops involved numerous support personnel, without whose efforts we could not have managed. In particular, we are grateful for the tireless services of Susan Pond who handled most of the telephone contacts with our workshop participants and supported the project in countless ways from its inception through the preparation of this report.

Finally, we would like to acknowledge our indebtedness to our esteemed colleague and friend Donald A. Waterman, who died January 4, 1987, after a serious illness. Don conceived this project, bringing it to life with insight and direction blended with joyful excitement. The project is a tribute to his talent for securing the collaboration of practitioners in the field of expert systems by virtue of the professionalism and candor that accompanied his far-reaching personal contacts and of his preeminence in the design, characterization, and evaluation of expert system tools. We are fortunate to have had the pleasure of working with Don, whose friendship, perspicacity, elegance, humor, and compassion brightened the lives of all who knew him.

CONTENTS

FIGURES

TABLE

I. OVERVIEW

Expert systems represent a new approach to solving complex problems with computers, i.e., by encoding human expertise, using principles developed in artificial intelligence (AI).[1] The approach differs sharply from traditional programming and system design in that it focuses on the explicit representation of knowledge and on mechanisms for manipulating and drawing inferences from that knowledge. Expert systems therefore require new system analysis techniques (referred to as "knowledge engineering") to extract knowledge from human experts, and new programming techniques to represent and use this knowledge. This new paradigm has spawned a host of "expert system building tools."

The plethora of such tools now on the market and under development can be very confusing to a system developer or project manager who must decide which (if any) to use for building a new expert system. The tools are typically large, complex systems in themselves, requiring major investments of time, money, and effort to yield their full advantage. Applied expert system technology is still in its infancy, and the concepts and techniques required to build expert systems are still fluid. Although this rapid evolution bespeaks healthy technological progress, it also produces unavoidable confusion. Even developers with considerable experience in building traditional systems may feel adrift in the sea of new ideas and jargon associated with the new paradigm. It is therefore important to develop guidelines for evaluating and choosing expert system tools.

Many recent studies have addressed the evaluation of expert system tools and technology. These studies fall into four broad categories: catalogs of tools, feature comparisons across large numbers of tools, detailed comparisons of a small number of tools that fall into the same class (e.g., personal computer tools), and comparisons of tools with respect to an application area. Bundy (1986) and Waterman (1986a) developed extensive catalogs of expert systems and techniques. Gilmore and Howard (1986) and Harmon and King (1985) performed feature comparisons across large numbers of tools. Beach (1986) and Richer (1986) compared in detail four expert system tools: KEE™ ART™, S.1, and Knowledge Craft. Examples of evaluations relevant to particular application areas include Culbert's evaluation (1986b) of

[1]Waterman (1986a) and Hayes-Roth, Waterman, and Lenat (1983) describe the nature of expert systems and how such systems are built.

six tools with respect to the needs of the National Aeronautics and Space Administration (NASA) and the evaluation procedure of Mayer et al. (1986) for selecting the right tools for manufacturing problems. (An annotated bibliography of these and other studies is given in the Appendix.)

Our primary goal in this study was to develop a framework of criteria and a methodology for selecting an expert system tool for a particular problem. Such a framework is only a starting point, however; it must evolve as the field and the tools mature. Nevertheless, our methodology should prove general enough to apply to any problem and any set of potential tools for the foreseeable future (which, we realize, is rarely very long in computer science).

It is important not to confuse the tool used to build an expert system with the expert system itself. As shown in Fig. 1, the tool includes both (1) the language for representing and accessing the system's knowledge and (2) the support environment. Confusion often arises because the support environment is also part of the completed expert system. We are concerned here only with evaluating expert system tools, not the expert systems they have been used to build. Another source of confusion is the term "user." It is important to distinguish between the user of the expert system tool and the end-user of the expert system.

The goal of expert system tool evaluation is to determine which tool characteristics are necessary and best suited for accomplishing a given task. Therefore, it is really a selection task: matching an expert system tool to its intended use.

Our framework is adaptable, and we expect it to fit virtually any domain, problem, and set of available tools. Our methodology applies this framework to select the most appropriate tool (or tools) for a given task. The process of tool selection can never be entirely mechanical—the skills and knowledge of the expert system developer will have a significant effect on the outcome. However, the framework and methodology presented here provide a sound point of departure.

In addition, our results should be of use to expert system tool designers and researchers. Our framework represents those criteria that are relevant to selecting an expert system tool; applying this framework reveals the strengths and weaknesses of individual tools and of the current generation of tools in general. It therefore serves as an indicator of the maturity of the field and as a guide for designing improved tools and choosing promising areas for research.

In the course of this study, we identified and explored many issues of concern to both tool developers and users. These are elucidated in detail in later sections. Among the most emphatic of these is the reali-

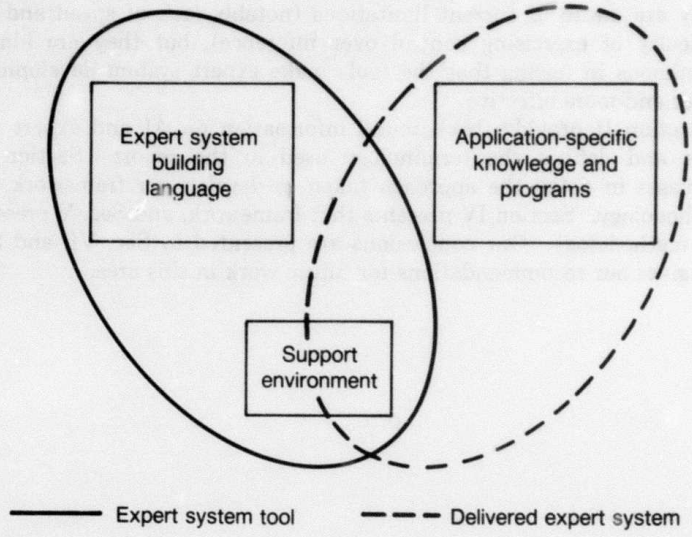

Fig. 1—Expert system tools and expert systems

zation that expert system development is first and foremost *software engineering*, a fact that is reflected in the importance of such issues as integration, portability, fielding, maintainability, debugging support, and documentation. Expert system tools may be viewed as very high-level programming languages, fitting above languages such as LISP and Pascal in the language hierarchy (i.e., machine languages, assembly languages, high-level languages). Criteria that are appropriate for evaluating programming languages and other software are therefore often applicable to expert system tools as well.

The relationship between expert system development and software engineering is symbiotic. The new paradigm has already made at least two significant contributions to the traditional software engineering discipline. First, it has shown the way to encode knowledge explicitly and declaratively rather than implicitly and procedurally, thereby making programs more understandable and maintainable. Second, it has pioneered the new software development strategy of prototyping and refinement, as opposed to the traditional cycle of specification, design, and implementation.

Users are overwhelmingly convinced of the value of these tools. They are aware of current limitations (notably lack of speed and the difficulty of exercising control over inference), but they are almost unanimous in feeling that the tools make expert system development easier and more effective.

Section II provides background information on AI and expert systems and defines the terminology used in the report. Section III discusses in detail the approach taken to develop our framework and methodology. Section IV presents that framework, and Sec. V presents the methodology. Our conclusions are presented in Sec. VI, and Sec. VII gives our recommendations for future work in this area.

II. EXPERT SYSTEMS AND EXPERT SYSTEM TOOLS

This section provides background on AI, expert systems, expert system tools, and the role of such tools in the development of expert systems.

A NOTE ON TERMINOLOGY

The problem of terminological ambiguity, common in computer science in general, is especially serious in the area of expert systems. For this report, we have adopted a consistent nomenclature, and we indicate desired meanings and connotations where necessary. For example, the term *expert system* means a system built using a knowledge-based approach to software development that applies expert knowledge to solve difficult real-world problems, as shown in Fig. 2 (Waterman, 1986a). To be strictly accurate, we should call the tools for building such systems *knowledge-based system building tools*. However, we use *expert system* and *knowledge-based system* interchangeably to refer to the general class of systems applying knowledge-based techniques.

For simplicity, we use *tool* (or, equivalently, *expert system tool*) to mean any piece of software intended to help design, build, deliver, or maintain an expert system.[1] A tool comprises not only a specific software environment but all aspects of the software entity and its use, including training, documentation, ease of use, vendor support, and cost.

The *development environment* is the environment in which a tool is used to design and implement an expert system. The expert system developed with a tool and delivered to its end-users is referred to as the *target (expert) system*, and the environment in which it runs is referred to as the *target* or *delivery environment*.

The term *developer* is itself ambiguous in the context of this report, since it can refer to either the developer of a tool or the user of a tool who is the developer of a target system. We therefore qualify it whenever ambiguity might arise. We use the term *vendor* to mean a commercial developer/vendor of an expert system tool.

[1]These tools are sometimes referred to in the literature as "expert system shells," but we do not use that term here.

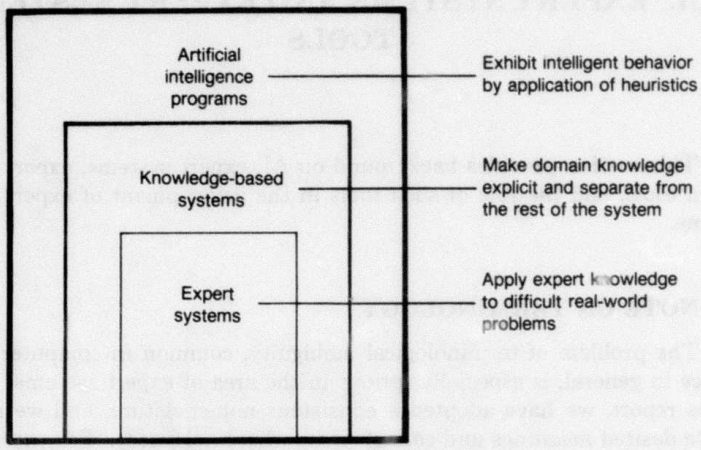

Fig. 2—The relationship between knowledge-based systems and expert systems

The term *user* is also ambiguous, since it can refer to either the user of a tool or the user of an expert system produced with a tool. Therefore, in this study, *user* means the user of a tool, and *end-user* means the user of the target expert system produced with a tool.

Application characteristics[2] include all aspects of the problem domain, the chosen problem within that domain, and the development environment, team, and project that implement the target expert system. These characteristics form the following hierarchy:

Problem characteristics
- Problem domain
- Problem to be solved within the domain
- Knowledge acquisition/expertise
- Target environment

Project characteristics
- Scope
- Development environment
- Development team

[2]In earlier presentations and in the workshops, we used the term *problem characterization* instead of *application characteristics*. We have since generalized the concept and changed the terminology accordingly.

Together, they characterize and provide requirements and constraints for the target system and the process by which it will be built. *Problem characteristics* refer to all aspects of the problem domain and the chosen problem, and *project characteristics* refer to all aspects of the development project, team, and environment. The term *problem domain* (or simply *domain*) is used in its conventional sense, meaning an area of expertise containing problems of interest.

We distinguish the *features* of a tool from the *capabilities* provided (or supported) by those features. A tool's features are the particular mechanisms it provides for doing things, whereas its capabilities are the things it can do. Features are tool-specific means to the ends of providing capabilities. For example, a tool may support the *capability* of representing object-type hierarchies by means of different *features*, such as frames or rules. (A given feature may support different or multiple capabilities at different times or in different tools; similarly, a given capability may be provided by different or multiple features at different times or in different tools.)

We discuss a number of *metrics* to be applied to tools, using this term in the most general sense. Metrics may be qualitative measures, rather than quantitative: applying a qualitative metric produces descriptive text rather than a number.

There are various ways of actually evaluating or comparing tools using a given metric. We suggest and discuss a number of *assessment techniques*[3] for applying metrics to tools.

Our extensible 5-dimensional *framework* of evaluation criteria is intended to be augmented, refined, and culled to suit the purposes of a particular tool selection process for a particular project.

Finally, we use the term *methodology* in the conventional, albeit imprecise, sense of a particular method rather than a science or system of methods. Our methodology enables refining and applying our framework to a real problem and a set of available tools, in order to choose the appropriate tool for that problem.

APPLIED AI: THE EXPERT SYSTEM PARADIGM

Artificial intelligence is a research area that began soon after the birth of computer science. As soon as computers were recognized as devices for manipulating symbols, rather than simply computing numbers, it was realized that they had the potential to perform tasks

[3]In earlier presentations and in the workshops, we used the term *methods* instead of *assessment techniques*. We have changed the term to avoid confusion with the term *methodology*.

that are normally thought of as requiring intelligence. An early (though in retrospect, naive) optimism developed, leading to predictions about automatic translation of Russian to English, voice-driven typewriters, automated theorem provers, and computer chess champions. However, it soon became clear that such problems are extremely difficult and do not submit to quick and clever solutions. They require in-depth understanding of the domain and specialized problem solving strategies that humans develop only after long experience. AI therefore entered a period of retrenchment during which the emphasis shifted from trying to build intelligent machines to trying to formalize the kinds of knowledge and problem-solving abilities humans apply in various domains.

The expert system paradigm essentially involves encoding expert knowledge (facts, rules, strategies, etc.) in a "knowledge base" that can be read by humans and used by a computer. Representing knowledge in a human-readable form allows experts to verify and revise the knowledge to improve the system's behavior, and it allows the resulting expert system to interact with experts (and nonexpert end-users), using terminology and concepts from the domain itself. The knowledge must also be represented in a form that allows a general-purpose program (referred to as an *inference engine*) to derive conclusions from the knowledge base. This can enable the expert system to solve problems whose solutions were never explicitly programmed.

EXPERT SYSTEM DEVELOPMENT

Knowledge-based expert systems provide a means of solving problems for which algorithmic and formal solution approaches do not currently exist (Paul, Waterman, and Peterson, 1986). These systems employ elaborate reasoning about abstract concepts in complex domains (Shortliffe, 1976; Van Horn, 1986). Expert systems are distinguished from conventional software by (1) their reliance on an explicit knowledge base and (2) the separation of that knowledge from the inference engine that interprets it. The overall structure of a typical expert system is shown in Fig. 3 (Waterman, 1986a).

The problem-solving power of an expert system program comes from both the knowledge it possesses and the formalisms and inference schemes it employs. Domain knowledge is encoded separately from program control and general problem-solving strategies. Waterman (1986a) provides a detailed description of expert system technology and numerous examples of its application.

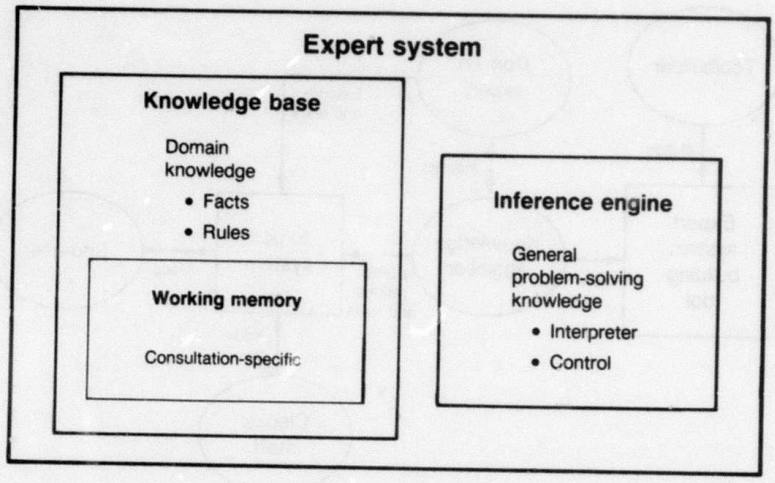

Fig. 3—Expert system structure

The process of building an expert system, called *knowledge engineering*, typically involves interaction between a system builder (the *knowledge engineer*) and a human expert. Figure 4 (Waterman, 1986a) shows the roles and interactions of the people involved in building an expert system. Although the term *end-user* usually refers to the ultimate consumer of the system's services, it refers here to anyone who uses the expert system for any purpose, including the domain expert, the knowledge engineer, and the clerical staff.

The knowledge engineer interviews domain experts to collect knowledge, organizes this knowledge, and determines how it will be represented, using the tool's formalism. The acknowledged primary bottleneck in the construction of an expert system is the transfer of this information into the knowledge base (Buchanan et al., 1983; Hendrix, 1980; Hayes-Roth, Waterman, and Lenat, 1983). Drastal (1984) noted that building a major expert system application in fewer than five person-years seemed an elusive goal: "The curve of development effort has leveled off around five person-years for all but the simplest consultation systems . . . . Knowledge acquisition remains a bottleneck on the way to passing the five-year barrier."

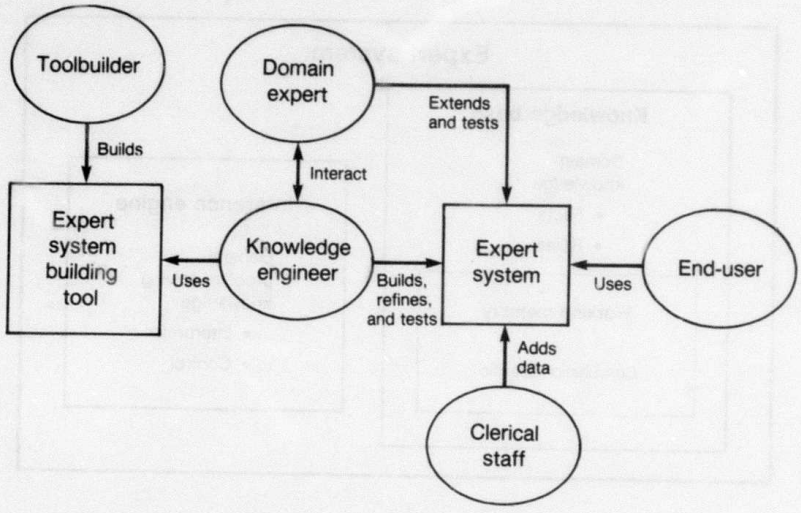

Fig. 4—People involved with expert systems

Expert system tools are software environments used for constructing knowledge-based systems. They include very high-level programming languages called *knowledge engineering languages* and an assortment of support facilities such as debugging aids, explanation facilities, run-time data acquisition facilities, and knowledge base editors.

Knowledge engineering languages provided by expert system tools differ from conventional programming languages in that they provide convenient ways to represent and organize complex, high-level concepts from the domain. *Knowledge representation* is concerned with how knowledge is structured in an expert system. Barr and Feigenbaum (1981) summarize the standard set of techniques which alone or in conjunction with others are supported by a given tool. The representations most widely used in current expert system tools are *rules* and *frames*.

It is necessary to examine tool features, such as the representation paradigms supported, in terms of their functionality rather than their surface appearance. Although expert system practitioners agree on some very general definitions of these terms and concepts, there are many shades of meaning and variations in their specific application and interpretation. For example, in KEE, frames are called *units*,

properties of units are called *slots*, and properties of slots are called *facets*. In S.1, on the other hand, frames are called *classes*, properties of classes are called *attributes*, and properties of attributes are called *slots*. Similarly, a *rule* in ROSIE is not the same as a *rule* in ART, and both of these are different from a *rule* in RuleMaster™. An illustrative rule from each system is shown in Fig. 5 (see Fain et al. (1981), Clayton (1986), and Radian (1986a) for details). Even at this very basic level, the problem of ambiguous terminology is acute. The presence of a particular term in a tool's documentation does not necessarily imply that the tool supports the representational paradigm normally denoted by that term.

EXPERT SYSTEM TOOL EVALUATION

In contrast to other evaluation studies, we adopt a very encompassing view of an expert system tool. That is, we use the term *tool* to apply both to the specific software environment and more generally to all aspects of the software entity and its use. Tool characteristics include training, documentation, ease of use, vendor support, and cost. This broad viewpoint is necessary to evaluate tools in a meaningful context; for example, for a system with anticipated longevity, the viability of the company supporting a tool is equally as important as the concrete functionality of the tool. In the course of this study, we attended vendor training courses and also got hands-on experience with the tools so that we could better understand issues of tool complexity, ease of learning, quality of training, etc.

Expert system tools are designed to help the system developer (i.e., the knowledge engineer) represent knowledge and get an initial version of an expert system working quickly, so that the domain expert can observe its behavior and suggest revisions or modifications. A number of popular representations are available, and different tools provide different alternatives (e.g., frames, rules, multiple worlds). The tools also vary widely in programming support environments, graphics capabilities, underlying programming languages, machines they can run on, etc. Since the tools are used to develop software, they must also be judged in terms of the performance, robustness, integrability, and portability of the software they produce.

These factors, coupled with an order-of-magnitude price range, make the selection of a tool a critical decision. A tool that is appropriate for a given task can be of great help in producing an expert system, whereas an inappropriate tool may incur unnecessary costs in money, learning time, and lost productivity. While selecting a tool obviously

```
Match the message:
        {"Hello"}       display HOWDY;
        {"Goodbye"}     display SOLONG and logout;
        {something}     display WHAT?;
        Default:        display SAY SOMETHING!.
```

A ROSIE rule (Fain et al., 1981)

```
(defrule denise-and-denephew
    (brother-of ?father ?uncle)
    (child-of   ?father ?child)
    (OR (AND (sex-of ?child female)
             (=> (assert (niece-of ?uncle ?child))))
        (AND (sex-of ?child male)
             (=> (assert (nephew-of ?uncle ?child)))))
=>
    (assert (uncle-of ?child ?uncle)))
```

An ART rule (Clayton, 1986)

```
IF degrees < 30 IS
    "T":  ("cold" -> returnval, goal)
    ELSE IF degrees < 60 IS
        "T": ("cool" -> returnval, goal)
        ELSE ("warm" -> returnval, goal)
```

A RuleMaster rule (Radian, 1986b)

Fig. 5—Examples of ROSIE, ART, and RuleMaster rules

requires the ability to evaluate tools, such evaluation cannot be performed in the abstract. It is conditioned by the characteristics of the domain and the chosen problem within that domain, as well as the *development context*, which includes the scope and purpose of the expert system being developed and whether it is to be a prototype, proof of concept, exploratory development, or delivered, supported software product. Tool selection is also conditioned by factors such as cost; suitability of tools for the available machines; ability to integrate with existing hardware, software, or databases; and familiarity with the tools themselves, their underlying programming languages, or their representational paradigms.

PROJECT BACKGROUND

This study was a natural outgrowth of RAND projects that included the development of the RITA (Anderson, 1976, 1977) and ROSIE (Sowizral, 1985; and Kipps, 1986) languages for rule-based programming, a number of expert systems (Waterman, 1981; Callero, 1984; Paul, 1986), and an explanation facility implemented in ROSIE (Waterman, 1986b). One of the earliest attempts to compare and evaluate tools for building expert systems was an expert systems workshop held in San Diego, California, in August 1980. At that workshop, eight 2-person teams, each having expertise in the use of a different expert system tool, spent three days implementing eight different prototype expert systems to address the same problem, the handling of chemical spills at Oak Ridge National Laboratory. This workshop was supported jointly by the National Science Foundation (NSF) and the Department of Defense's Advanced Research Projects Agency (ARPA), subsequently renamed the Defense Advanced Research Projects Agency (DARPA). The comparison of the tools employed at the workshop and the paradigmatic problem developed there (usually referred to as "the spills problem") have since been used frequently for comparison and training (see Waterman, Hayes-Roth, 1982; and Hayes-Roth, Waterman, and Lenat, 1983).

III. APPROACH USED IN THIS STUDY

This section discusses the approach we took in developing our ideas and producing our framework and methodology. Basically, we collected and analyzed information to identify issues; synthesized new ideas about tool evaluation; and refined and validated our ideas by means of peer exposure.

COLLECTING INFORMATION

To collect information relevant to the evaluation and selection of expert system tools, we surveyed expert system building activities, reviewed commercial and research expert system tools, conducted in-depth studies of particular tools, and sponsored two workshops. These activities also refined and validated our ideas.

Survey Overviews

We examined more than 200 expert systems and over 100 expert system tools to identify both common and unique characteristics. We also surveyed evaluations by other researchers (see the Appendix). These evaluations fall into four broad categories: catalogs of tools; feature comparisons across large numbers of tools; detailed comparisons of a small number of tools that fall into the same class (e.g., personal computer tools); and comparisons of tools with respect to an application area.

Catalogs of Tools. Bundy (1986) cataloged over 250 entries of software products and AI techniques. Waterman (1986a) describes over 95 tools and offers insight into the understanding and selection of tools. Walker and Miller (1986) describe more than 70 tools; their catalog is updated yearly.

Feature Comparisons. Feature comparisons define relevant features or criteria for distinguishing among tools. Gevarter (1986) first identified features that are important for solving certain types of problems (e.g., forward chaining is important for solving a monitoring problem) and then compared 20 tools with respect to tool features and problem types. Gilmore and Howard (1986) identified important criteria for selection of both large- and small-scale tools. They developed detailed knowledge representation and inferencing features for distinguishing among the large-scale tools. Harmon and King (1985)

categorized tools as small, large-and-narrow, or large-and-hybrid. They then evaluated 17 tools with respect to knowledge representation and inference and control strategies and identified the best tools for particular problem types.

Detailed Comparisons. The monthly newsletter *Expert Systems Strategies* reviews expert systems by category. The volume addressing personal computer tools (September 1985) evaluates eight tools across a number of characteristics and presents a technique for deriving the value of small tools for practical business applications on the IBM-PC™. Beach (1986) compared KEE, ART, S.1, and Knowledge Craft by implementing the same prototype problem with each tool. This led to the identification of several critical tool features. Richer (1986) also compared KEE, ART, S.1, and Knowledge Craft and developed five main criteria for evaluating them: basic features, development environment, functionality, support, and cost.

Comparisons Within an Application Area. Culbert (1986b) determined the tool features most critical to NASA expert system development projects and evaluated 16 tools with respect to those features. Meyer et al. (1986) defined expert system tool capabilities required by the specific needs of manufacturing applications and identified shortcomings of existing tools. They also discussed issues to be considered in the decision to acquire an expert system tool and in the selection of the proper tool. Wall et al. (1985) compared KEE, ART, and Knowledge Craft with respect to the characteristics critical to a Navy command and control system by implementing a prototype problem dealing with monitoring the readiness of the Navy's Pacific Fleet.

These surveys established the appropriate context for expert system evaluation and provided a baseline for our research. We determined the primary need to be the identification of evaluation criteria and the techniques for applying these criteria. Such criteria are also useful for deciding how to enhance existing tools or design new ones.

Vendor Training

To expand our knowledge and understanding of the experiences and difficulties faced by tool users in attending training courses, using documentation, and becoming proficient in tool use, we took several vendor training courses. We received the standard documentation and training materials and had ample opportunity to exchange views with other attendees. This training provided useful insights and experience on how readily a particular tool can be effectively employed by the user. This training, ranging from informal one-on-one sessions to two-week formal training courses, pointed out effective techniques, areas of general confusion, and many related issues.

Workshops

To get representative viewpoints of tool developers and a broad sampling of tool users, we held two workshops at RAND. At the first workshop, a group of commercial tool builders focused on identifying important issues to be considered in tool evaluation. We presented our evaluation framework, and the group refined and formalized ideas embodied in that framework. At the second workshop, a group of developers who were actively involved in building expert systems described their experiences with different tools and different-sized applications. The discussions and results of these workshops are reported in detail in companion RAND Note N-2603-DARPA, *Evaluating Expert System Tools: A Framework and Methodology—Workshops.*

We also attended two workshops sponsored by the Defense Advanced Research Projects Agency (DARPA) and the University of Ohio (October 1986) for expert system tool researchers and developers.

ANALYSIS AND SYNTHESIS: DIMENSIONS OF EVALUATION

Expert system tools cannot be evaluated simply by listing features and attempting to compare them in checklist fashion. Feature comparisons are limited by the depth of meaning behind the labels assigned to features and the tendency to focus on superficial aspects. It is necessary to address questions such as, How is quality of documentation assessed? How does one compare features of tools? How large an application can be managed? For example, numbers of rules do not necessarily reflect the size of the application that can be managed; differences in the functionality and granularity of rules can make such comparisons all but meaningless. Furthermore, enumerations of current features (e.g., backward chaining) may be of little value for future systems. Therefore, we focused on the capabilities various features provide (e.g., goal-directed reasoning). Such capabilities are themselves continually evolving, but they are at a higher semantic level than features. Evaluating tools in terms of capabilities instead of features therefore avoids some terminological ambiguity and should be less prone to early obsolescence.

To synthesize a framework of criteria characterizing the issues involved, we identified five dimensions that define tool evaluation: application characteristics, tool capabilities, metrics, assessment techniques, and contexts. These dimensions are discussed in detail in Sec. IV.

Some earlier tool evaluation efforts (e.g., Beach (1986), Gilmore (1986), Harmon (1985), Harmon and King (1985), Richer (1986), Ruby (1986)) evaluated tools in the abstract. While this approach shows how the tools differ, it is not well suited to choosing a tool for a particular application, since it does not focus on application-specific aspects and interdependencies of the tool and its environment. This type of evaluation also becomes quickly outdated; tool developers can release new versions of tools faster than studies can be performed and published. Many studies develop lists of tool features or capabilities that are relevant to a particular application or class of applications and then rate how well each tool supports each needed capability (Beach, Culbert, Gevarter, Mayer, Wall). This corresponds to the cross-product of two dimensions of our framework: matching application characteristics with tool capabilities.

Although our framework shares many elements with these studies, it is unique in several ways. In particular, it adds three new dimensions: metrics, assessment techniques, and contexts. Rather than simply comparing features in the abstract, we recommend specific assessment techniques (e.g., questionnaires or case studies) to evaluate specific metrics (e.g., flexibility or efficiency) of tool capabilities in particular contexts (e.g., during development), given the relevant application characteristics. Although these additional dimensions make evaluation more complex, we believe they also make it more meaningful and therefore more useful.

Our framework organizes and structures the criteria we have identified. It also provides a medium for making ideas and insights more concrete and formal, enabling such information to be shared within a diverse community.

IV. THE TOOL EVALUATION FRAMEWORK

Evaluating and choosing a tool demands a broader context than is usually suggested. "Matching a tool to a problem" should be expanded to include all aspects of the problem domain, the problem itself, and the anticipated project—that is, "matching a tool to *its intended use*."

Our framework and its associated methodology have been developed to apply both to existing tools and to new tools. Our dimensions are therefore not confined to those characteristics for which systems exist; for example, we allow for tool capabilities that are not present in any current tools. Similarly, the framework and methodology are intended to be applicable to a number of different tasks, including evaluating and selecting existing and newly developed tools; comparing tools; designing tool enhancements; suggesting future-generation tools; and focusing research in expert system technology.

Although the framework includes many criteria, those criteria can be pruned by identifying items of particular significance to a project. Furthermore, most of the dimensions can be prioritized. Pruning and prioritizing should make evaluation manageable. This is discussed further in Sec. V.

FRAMEWORK OVERVIEW

We view tool evaluation as involving five distinct dimensions:

- *Application characteristics*: the problem, its domain, and the project.
- *Tool capabilities*: the functionality of the tools being considered.
- *Metrics*: measures for evaluating the capabilities of tools.
- *Assessment techniques*: ways of applying metrics.
- *Contexts*: the different requirements for using a tool in different phases of expert system development.

These five dimensions are used in the following way to evaluate a tool for building a given application or range of applications: Given the relevant *application characteristics*, apply *metrics* by means of *assessment techniques* to evaluate particular *capabilities* of a tool in particular *contexts*.

A relatively concrete example would be:

> *A two-person-year project is proposed to develop an expert system to locate spills in a chemical plant. Application characteristics include the size and experience of the development team, their computing environment and funding level, the expected end-users and delivery environment, and all relevant problem and problem domain constraints. Evaluate the flexibility of tool X's knowledge acquisition capabilities in the development context, by analyzing case studies.*

The following schema, designed to be read as a sentence, summarizes the use of the framework dimensions:

Use	an *assessment technique*
to evaluate	a *metric*
of a tool's	*capability*
in	a *context*
given	the *application characteristics*.

These dimensions are discussed in detail below. For each dimension, we present a rationale for the criteria to be considered. These criteria represent only a starting point; they must be scrutinized, pruned, and extended as appropriate for a given evaluation task.

One particularly crucial concept, integration, does not fit neatly into this framework. Integration can be viewed as an application characteristic (e.g., requiring integration of a tool into the development environment or of the target expert system into the delivery environment), a tool capability (e.g., supporting the integration of expert systems into existing environments), or a metric to be applied to other tool capabilities (e.g., measuring how well a tool's representational paradigms are integrated). Integration is a multifaceted concern that generates many constraints for tool evaluation and expert system development.

APPLICATION CHARACTERISTICS

Application characteristics represent the impact of the application on tool evaluation. Although the problem domain and problem may be ill-defined, it is crucial to determine as many characteristics as possible, to derive appropriate requirements and constraints. To some extent this dimension overlaps traditional requirements analysis, though we focus on the following aspects, which are either unique or particularly crucial to expert system development:

Problem characteristics
- **Problem domain**
  - Kinds of knowledge
  - Constraints
- **Problem to be solved within the domain**
  - Special processing/knowledge/representation
  - Problem type
  - Other problem attributes
- **Knowledge acquisition/expertise**
  - Characteristics and constraints
- **Target environment**
  - Constraints
  - End-users

Project characteristics
- **Scope**
  - Goals and budget
- **Development environment**
  - Constraints
- **Development team**
  - Characteristics

Problem Characteristics

Developing an expert system involves many tasks beyond analyzing the problem and solving it. They include identifying experts and other sources of expertise in the domain; acquiring and validating the necessary knowledge; identifying and understanding the end-users and their environment; and organizing and managing the development effort. The tool chosen can have a dramatic effect on each of these tasks. To give a concrete example, Fig. 6 shows some problem characteristics of an illustrative system, NEOMYCIN (Clancey, 1981; Hasling, Clancey, and Rennels, 1983).

Many of the requirements for a target expert system imply requirements for the tool used in building it. For example, end-user interface requirements logically affect only the target system, but the ability to implement a given interface for that system may be sharply constrained by the tool used. Similarly, portability of the target system or its ability to integrate with existing databases, software, or hardware may dictate requirements for the tool. Though many of the issues identified in this section are largely common sense and should be obvious to anyone with extensive design experience, they are included here for completeness, and as an aid for readers lacking such experience.

Problem domain

Infectious blood diseases

Medical knowledge: relating infecting organisms with patient history, symptoms, and laboratory test results; drug treatment procedures; diagnostic problem-solving methods.

Problem to be solved within the domain

Diagnosis of and therapy recommendation for hospital patients with bacteremia, meningitis, and cystitis infections. Also intended to use knowledge-base for teaching medical students.

Knowledge sources

Existing MYCIN knowledge base; medical practitioners/diagnosticians; medical textbooks.

Target environment

End-users: physicians, medical students

Fig. 6—Problem characteristics of the NEOMYCIN system

Well-formulated problems can be handled with traditional system analysis and algorithmic approaches, but few expert system development efforts are well formulated in advance. A primary reason for using the expert system approach is to find a way to handle problems that are not well understood. Expert system technology comes into its own in situations where the problem domain is complex, domain knowledge and expertise are ambiguous, and the problem itself can only be understood after considerable exploration and experimentation, possibly involving several prototyping and design iterations.

This produces a chicken-and-egg problem: an appropriate tool can be chosen only after the requirements imposed by the problem are understood, but understanding these requirements may only be possible after experimenting with techniques and building a prototype expert system, which requires choosing a tool. This suggests a strategy of choosing an initial tool for experimental purposes and then choosing a (possibly different) tool based on what is learned from prototyping. Choosing different tools for prototyping and development might also result from different requirements for these phases of a project, rather than from a change of direction produced by the refinement of requirements during the prototype phase. This is discussed further below. It

is also possible that no single tool will handle all aspects of a problem. Of course, the investment (in both money and learning time) required to use multiple tools, whether in sequence or in concert, makes it impractical in many cases.

Expert system tools may also be evaluated without a specific problem—or even a specific domain—in mind. Many organizations are exploring this technology simply to understand it and to see whether it is applicable to any relevant problems. In some cases, a domain or range of domains may be broadly defined, but in others, the point may be to determine which domains such a tool might make tractable. In these cases, the domain and problem characteristics may be unknowns whose values become determined (though possibly still not obvious) by the selection of a particular tool.

Problem Domain. The kinds of knowledge and processing that characterize a domain may provide useful criteria for choosing among tools. For example, applications in earth science or military simulation may require spatial reasoning; financial or legal applications may require strict accountability; and process control, nuclear power, or space systems may have real-time and critical reliability requirements.

In some cases, a tool may incorporate specific mechanisms and knowledge oriented toward a particular domain. For example, a tool for building financial expert systems might incorporate concepts and computational techniques such as tax tables or amortization algorithms. Similarly, some existing tools are tailored toward process control and real-time data acquisition, whereas some provide automated rule induction for domains where expertise consists of examples rather than explicit rules. As experience with expert systems grows, tools may emerge with utility packages or knowledge bases tailored to specific domains. Prepackaged domain knowledge of this kind may become an important consideration in choosing a tool, just as the availability of subroutine libraries for statistics, graphics, or database access is often a consideration in choosing a programming language. Domain-related tool capabilities are likely to become more widespread as specialized techniques are developed for different domains, and tools are differentiated into "vertical market" niches.

Problem to Be Solved Within the Domain. The particular problem to be solved may involve special kinds of knowledge, processing, or representational requirements that may lead to special requirements for a tool. The problem may also establish requirements and constraints for the capacity, performance, and delivered cost of the target system, as well as for its availability, reliability, robustness, and maintainability.

Many problems require that a target expert system communicate and integrate with special databases, software, and hardware (e.g., sensors or effectors) in the target environment. These integration issues are often crucial because of the difficulty of building such interfaces.

It is also tempting to try to characterize a problem according to its "type" (e.g., diagnosis, classification, planning, etc.), although most problems consist of combinations of such types. For example, many expert system tools evolved from early research in medical diagnosis, so there has been a tendency to think of diagnosis as a coherent problem type, spanning many domains. Whether this is more than superficially true is an open question. Nevertheless, an appropriate taxonomy of primitive types (analogous to diagnosis, though possibly at a different semantic level) might prove useful in selecting tools. An attempt to define such a taxonomy is currently under way in the research community (see, for example, Chandrasekaran, 1986), though results to date are tentative.

Additional problem attributes that should be considered include expected complexity and storage requirements; operational constraints such as execution speed, real-time requirements, compatibility requirements, physical environment, hardware portability requirements, or the need for formal verification or proof of correctness; and formal properties, such as decomposability, the degree to which the problem lends itself to algorithmic vs. heuristic solution, and symbolic or numeric requirements.

User-machine interaction is also important in terms of both the required tool interface for the expert system developer and the required tool support for building the end-user interface for the target expert system. The degree of autonomy of the target system (i.e., whether it is to be used as a decision aid or as an autonomous controller) also affects its end-user interface requirements.

Knowledge Acquisition/Expertise. It is important to identify special characteristics or constraints that may apply to knowledge acquisition or the sources of expertise in the application domain, including the need for multiple knowledge sources or coordination of multiple knowledge bases.

Target Environment. The target environment for an expert system determines its delivery hardware and its need to integrate with existing hardware, software, databases, and networks. It also establishes requirements and constraints for the capacity, performance, and delivered cost of the target system, as well as its availability, reliability, robustness, and maintainability. Similarly, the characteristics of the expected end-users of the target expert system (their level of experience with computers, their domain expertise, their educational

background, etc.) determine user interface and explanation requirements for the target system and therefore for the tool.

Project Characteristics

Project characteristics include characteristics of the expert system project, its development environment, and its development team.

Scope. The scope, goals, and budget of an expert system effort are obviously among the most important factors in determining what kind of system will be built, and therefore what kind of tool is required. In particular, the scope of a project will determine which phases are targeted (e.g., prototyping, development, etc.), as represented by the context dimension.

Although they vary widely in emphasis, most expert system tools provide support for the entire process of building and delivering expert systems. However, a given project may not require the full range of support, since it may focus on certain phases of the process. For example, if the task is to prototype a system, or to show the feasibility of a solution to a problem, or even to explore representations without producing a running system (as is often the case in research projects whose goal is simply to evaluate the new technology), then a tool might be chosen for maximum representational flexibility and ease of building prototypes, with no emphasis on large-scale development support or delivery to end-user environments. On the other hand, if a project has already defined its problem and already has a representation or even a knowledge base, then a tool might be chosen to launch a development and delivery effort, without concern for knowledge acquisition or prototyping. The phase or phases targeted may also interact with problem characteristics as well as project characteristics. For example, the prototyping phase focuses on sources of expertise and problem formulation, whereas the fielding phase is concerned with robustness, integration, performance, and end-user interfaces.

Closely tied to scope is the question of budget. Ideally, more powerful tools allow a given expenditure of effort and money to produce more results, but the tools themselves cost money, and there is a time cost in learning to use them effectively. Especially for the first project using a given tool, these costs limit the scope that can be undertaken within a given budget. Cost is often an overriding (or vetoing) factor in deciding on a tool. Well-funded projects tend to consider only the high-end tools, under the assumption that these will deliver the greatest power. Low-budget efforts tend to consider only low-end tools, either because these are literally the only ones affordable or because the presumably greater power of a more expensive tool is not expected

to pay for its greater cost. Projects having widely different budgets tend to have access to different computing environments, and this also filters the choice of available and appropriate tools.

Development Environment. The development environment delimits the hardware and software on which a tool must run as well as the network and database interfaces it must provide during development. These factors may be given, or a project may have some control over them (e.g., the freedom to choose hardware based on the preferred tools).

Development Team. The composition of the development team (whether preexisting or contemplated) must also be taken into account. If the team is large, the cost of bringing all the developers up the learning curve for a tool may be prohibitive (though synergistic effects may make the costs scale somewhat better than linearly with the size of the group). Similarly, the size of the team may generate requirements for a tool to support cooperative development, configuration management, shared databases, etc.

The background, preferences, and previous experience of the team are also important, and will become increasingly so as tools begin to accrue significant user communities. Though not always homogeneous across the team, mindset is often a determining factor in the choice of a tool. Finally, characteristics of the knowledge engineering team (such as computer sophistication) will affect the kinds of tool support required for knowledge acquisition.

TOOL CAPABILITIES

The tools currently on the market provide many features supporting a wide range of capabilities. In addition, our workshops identified a number of other desirable capabilities, some of which are provided to a certain extent by existing tools and others of which are merely wishful thinking.[1]

We focus on the capabilities of a tool, rather than the specific features it provides for achieving those capabilities. This emphasizes the functionality of a tool rather than the specific implementation of that functionality. Most other evaluation studies, in contrast, tend to emphasize features or mix features and capabilities without distinction. However, users often require a particular capability and do not care which features are used to provide it.

[1]These capabilities are discussed in the Wish List presented in the companion Note.

As expert system technology and its tools evolve, the list of relevant capabilities will also evolve and expand. *Tool capabilities* is therefore the most dynamic of our dimensions. Examples of current capabilities and representative features that support them are shown in Table 1. This list is not exhaustive, and the examples of supporting features are merely illustrative. Many other features supporting these capabilities can also be found in current tools.

The set of capabilities relevant to a given evaluation must be identified on the basis of the given application characteristics. This dimension can therefore normally be filled in only after analyzing the application characteristics of the task at hand.

Table 1

ILLUSTRATIVE CURRENT TOOL CAPABILITIES

Capability	Examples of Supporting Features
Arithmetic processing	Arithmetic operators, extended floating point
Certainty handling	Certainty factors, fuzzy logic
Concurrency	Distributed processing, parallel processing
Consistency checking	Knowledge base syntax checking
Documenting development	Assumption/rationale history, code/data annotation
Explanation	Execution trace, knowledge base browsing
Inference and control	Iteration, forward/backward chaining, inheritance
Integration	Calling other languages, interprocess calls
Internal access	Tool parameter setting functions, source code
Knowledge acquisition	Rule induction, model building aids
Knowledge base editing	Structure editors, graphic rule lattice
Life cycle	Tool support for target system life cycle support
Meta-knowledge	Rules controlling inference, self-organizing data
Optimization	Intelligent look-ahead, caching, rule compilation
Presentation (I/O)	Text, graphics, windows, forms, mouse
Representation	Rules, frames, procedures, objects, simulation

METRICS

Metrics are applied to particular capabilities of a tool using the assessment techniques described below. The following aggregated metrics capture most of the relevant qualities of a tool:

- Cost
- Flexibility
- Extensibility
- Clarity
- Efficiency
- Vendor support

Explicit metrics are a unique aspect of our framework that allows tools to be analyzed in depth by looking at different facets of their capabilities (such as flexibility, efficiency, or ease of use), rather than evaluating them according to a single measure.

Figure 7 suggests the important qualitative relationships for five of the six metrics (excluding cost, which behaves rather differently). The relationships shown are typical, although they may vary for different projects.

Cost includes hidden expenses such as costs of training and integration, as well as the purchase price and support costs of a tool. Resources consumed (and therefore costs) may be money, personpower, machinery, supplies, computation used, elapsed time, etc. The effects of cost are felt throughout the life cycle of a system, but for

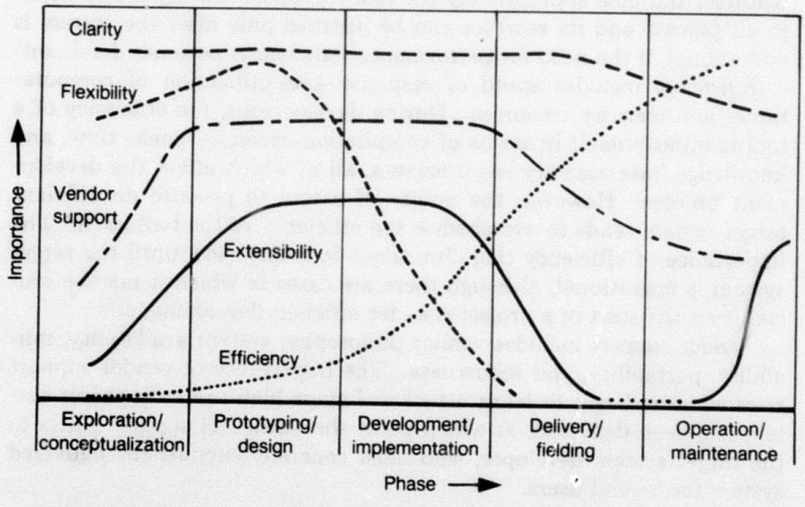

Fig. 7—Relative importance of metrics across development phases

evaluation purposes, its relevance peaks at the transitions between phases, where a project either commits to switching tools or stays with what it has. For this reason, cost is not shown in Fig. 7.

Flexibility includes representational power (data structures and reasoning mechanisms), adequacy to the given task or tasks, breadth of applicability, and sophistication. Flexibility may be antithetical to clarity (e.g., maintainability). It is most important in the initial stages of tool use, peaking during the prototyping phase. As implementation decisions are made, flexibility becomes less important and may actually become a negative factor as the need for maintainability leads to the requirement that the system be stable.

Extensibility includes breadth of applicability, access to system parameters, defeatability (i.e., the ease with which system parameters or functions can be overridden), ease of integration, portability, scalability, and "subsetability." Extensibility peaks during design and implementation and decreases in importance during fielding, by which time the system should be fairly stable. It may be important again later if maintenance requires new extensions or reintegration.

Clarity includes the ease of understanding and using a tool, cognitive efficiency (i.e., how many concepts must be kept in mind to use the tool), maintainability, modularity, learnability, coherence of the tool's features, and how appropriately the tool responds. Clarity is important in all phases, and its sacrifice can be justified only after the system is operational, if the need for performance (efficiency) becomes dominant.

Efficiency includes speed of response and utilization of computational and memory resources. During development, the efficiency of a tool manifests itself in terms of compilation speed, response time, and knowledge base memory requirements, all of which affect the development process. However, the ability of a tool to produce an efficient target system tends to overshadow the efficiency of the tool itself. The importance of efficiency therefore tends to remain low until the target system is operational, although there are cases in which it may be crucial from the start of a project (i.e., for efficient development).

Vendor support includes vendor philosophy, system availability, reliability, portability, and robustness. The importance of vendor support rises as users begin to learn a tool and stays high until the target system has been delivered, at which point the burden of support shifts to the target system developer, who must generally support the delivered system for its end-users.

ASSESSMENT TECHNIQUES

For some metrics, assessment techniques seem obvious and straightforward. Evaluating the initial cost of a tool, for example, is simply a matter of asking for a quote; however, even in this highly quantitative realm, evaluating learning cost, long-term cost, or cost-effectiveness is often far from trivial. Similarly, feature comparison is often performed by simply asking whether a tool has a given feature. Vendor sales departments encourage this technique by publishing checklists of features for their products. Unfortunately, as pointed out above, this kind of comparison can be quite misleading.

Developers who must choose a tool often rely on anecdotal evidence from colleagues who have applicable experience. This is not surprising, since there are currently few objective ways of applying metrics to a tool prior to using it. Moreover, any measurement technique that claims to be objective is viewed with suspicion, especially if it is also quantitative. For example, performance benchmarks are considered by tool developers and users alike to be unreliable because they are too easily "faked" or distorted by implementing special shortcuts or comparing incommensurable items (e.g., showing relative speeds for processing "rules," where the granularity and power of a rule may vary widely among tools).

Our approach to the application of metrics is to suggest a number of assessment techniques that do not attempt to produce quantitative measures, but instead produce textual results. When a metric is applied using one of these techniques, the result will require intelligent interpretation. However, we feel that this is an advantage, rather than a disadvantage, since it gives the decisionmaker valuable information.

The assessment techniques that seem the most promising are:

- Direct comparisons
- Benchmarks
- Interviews, questionnaires, and personal advice
- Libraries of case studies and development efforts
- Knowledge-based systems for tool evaluation

Although some of these techniques are implicit in other evaluation studies, we believe it is important to discuss them explicitly.

Direct Comparisons

Direct comparisons among tools can be valuable if they focus on corresponding capabilities of tools and make explicit the ways in which these capabilities differ. Comparison with an abstract standard or with "conventional" AI approaches (e.g., using LISP or Prolog) may also be useful.

Benchmarks

Benchmarks are by far the most controversial of the techniques proposed here, in part due to the connotations of the word *benchmark*. It is not used here to mean a quantitative performance measure or a feature-based comparison, such as how many rules a tool can process per second. Rather, we use *benchmark* to mean a well-formulated statement of a more-or-less fragmentary problem. The solution of the benchmark (or the impossibility thereof) should be published for each tool under consideration, along with any relevant performance data or development statistics.

Benchmarks can be small (for example, testing whether a tool can represent a class hierarchy) or large (such as the spills problem described in Sec. II). Small benchmarks would probably have little data associated with them, i.e., the style of the published solution would constitute the result. Larger benchmarks might include performance data or statistics, such as the time required to implement a solution using a given tool, but the emphasis would always be on the content and style of the published solution, and the evaluator would always be responsible for interpreting any numeric data in terms of the quality of the solution.

Small benchmark problems can be used to compare specific capabilities of tools, provided they are interpreted on the basis of the style of their solutions rather than their performance. Implementation of larger benchmarks using a number of different tools may be warranted prior to a major tool commitment. To use benchmarks, it is necessary to formulate problems that are specific but do not require a particular implementation. Some illustrative benchmarks are given in the companion Note.

Additional Techniques

Interviews, questionnaires, and personal advice from other developers, tool users, and colleagues can provide useful information. A library of case studies and expert system development efforts would be

a valuable source of evaluation data if it could be (1) large enough to be representative, (2) kept up to date as tools evolve and experiences change, (3) accessible by a wide range of potential tool users, and (4) indexed according to application characteristics.

A knowledge-based system for tool evaluation could be used to recommend relevant criteria or to help synthesize a decision based on the results of evaluations using various criteria. Though a general-purpose system of this kind is unlikely to be feasible for some time, an organization engaged in choosing tools on a recurring basis in a relatively constrained domain may be able to incorporate some of its evaluation expertise into an expert system of this kind.

CONTEXTS

Each context in which a tool can be used is named for the development phase in which it is dominant, although a given context may apply across several development stages: For example, tool requirements for "fielding" may apply early in the conceptual design stage of a project. Representing these contexts as a separate dimension emphasizes this independence and allows the concerns of each context (and the resulting tool evaluation criteria) to be applied at all stages of development. The transitions between development phases can be just as important as the phases themselves. Delimiting contexts as a fixed set of points must not obscure the transitions between these points.

The contexts are:

- Conceptualization
- Prototyping
- Development
- Fielding
- Operation/maintenance

The relevant contexts for a given development effort are derived from the project scope and the problem to be solved.

Conceptualization emphasizes a tool's support for conceptualizing, formalizing, and decomposing a problem, and for identifying and organizing key concepts and defining the problem's scope. This may involve exploration of feasible approaches and representations, as well as conceptual design.

Prototyping emphasizes a tool's facilities for guiding rapid development, eliciting different approaches and representations, and quickly trying alternative implementations.

Development considers a tool as it is used to develop an expert system. The tool's support for software development (including debugging facilities, configuration management, etc.) is emphasized here.

Fielding emphasizes a tool's facilities for porting from the development environment to the delivery environment and for integration and end-user interface support.

Operation/maintenance emphasizes a tool's support for the performance, maintainability, and supportability of target expert systems in their delivery environments.

V. METHODOLOGY

The framework presented in Sec. IV can be applied to a particular evaluation task by means of the following steps:

1. Determine application characteristics.
2. Identify relevant contexts.
3. Derive relevant tool capabilities.
4. Identify discriminating metrics and assessment techniques.
5. Identify available tools.
6. Filter available tools to identify candidate tools.
7. Prune and prioritize each framework dimension.
8. Apply the framework schema to evaluate and select tools.

Many tool evaluation studies use an implicit methodology (such as "rate the tools with respect to these criteria, weight the criteria, and use this to choose a tool"). However, we believe that it is important to use an explicit and objective methodology to avoid prejudice.

EVALUATION CAVEATS

There are several questions that should be asked prior to performing an evaluation. If satisfactory answers cannot be found, the validity of the evaluation process may be questionable.

Does the evaluation have an identified consumer? Evaluations for use by managers, technical staff, or end-users will have different requirements. The eventual consumer of the evaluation should be identified at an early stage to avoid producing superfluous or misleading results.

Will the evaluation be impartial and objective? Even though impartiality may be relative to the needs of the consumer, intentional biases should be made explicit to avoid any misunderstandings.

For all but the most informal purposes, evaluation must be objective. Priorities, importance weights, and evaluation algorithms must be established in advance, preferably by domain experts or other disinterested parties who are not prejudiced as to the outcome of the selection process. Once weights and algorithms have been chosen, they should be inviolable unless they are found to be untenable, in which case they should be formally redefined or relaxed. An evaluation in which such initial decisions are never made or are relaxed whenever they become inconvenient or subjected to pressure is worthless.

How will the evaluation be monitored for accuracy and bias? In other words, who evaluates the evaluator? If no such monitoring is performed, the consumer should be made aware of the possibility of errors and biases.

Will the evaluation be timely? The evaluation process must be designed to deliver its results in time to be of use. Since tools evolve rapidly, the results of an evaluation may quickly become obsolete unless precautions are taken to update them. It is important to make explicit mention of versions and release dates of the tools evaluated, to give consumers of the evaluation an explicit basis for judging timeliness.

THE EIGHT STEPS OF EVALUATION

In this section, we examine the eight steps of tool evaluation and selection listed above.

1. Determine application characteristics. Even though the application domain and problem may be ill-defined, it is necessary to determine the application characteristics insofar as possible, since the required tool capabilities are largely derived from these characteristics. It is equally important to determine project characteristics at this point. This involves defining the goals, scope, and budget of the development effort, and characterizing the development team and development environment. The contexts to be targeted are derived from the scope of the project, whereas the development team and environment may impose other constraints on tool selection.

Application characteristics should be weighted as to certitude and importance. The scope and goals of a project will determine which characteristics represent obligatory requirements and which ones are negotiable.

2. Identify relevant contexts. The context or contexts targeted for a project supply the other major factor for determining required tool capabilities. It is particularly important here to be realistic about which phases of development are to be undertaken with the tool(s) to be chosen. Targeting only the initial exploratory or prototyping phases may lead to choosing a tool that encounters a dead end if the development process is extended further. On the other hand, targeting all phases when only some of them are likely to be undertaken will overconstrain the selection process.

3. Derive relevant tool capabilities. The tool capabilities are derived from the application characteristics and the relevant contexts. It is important at this stage to weight these capabilities along a scale

from *required* to *desired*, for use in filtering the available tools in Step 6. These weights are derived from the application characteristic weights determined in Step 1.

4. Identify discriminating metrics and assessment techniques. Certain metrics, such as cost, may have high discriminating (or vetoing) power in choosing a tool. If such metrics exist for a project, they should be identified, along with the best available assessment techniques for applying them at this stage (there may be better assessment techniques that are not available at this stage due to the cost or time required to apply them). If no such discriminating metrics are appropriate for a project, this step has no effect.

5. Identify available tools. This step may necessarily involve some implicit filtering, since it is difficult to find all available tools, but it is desirable to make as complete a survey as possible, filtering explicitly in Step 6 instead.

6. Filter available tools to identify candidate tools. Using the *required* capabilities derived in Step 3 and the discriminating metrics (if any) identified in Step 4, the available tools should be filtered to produce a set of candidate tools to be evaluated in further detail.

7. Prune and prioritize each framework dimension. Each dimension of the framework should be pruned to eliminate irrelevant or inapplicable criteria, and the remaining items should be prioritized or weighted. Attempting this prioritization earlier in the process would be difficult due to lack of focus. At this point, enough should be known about the application characteristics, contexts, capabilities, and available tools to prioritize each framework dimension separately. For example, if cost filtering has already resulted in a set of candidate tools whose costs are very similar, cost would be an ineffective metric at this stage.

In most cases, strict prioritization will be inappropriate. For example, cost might be a more important metric than efficiency, but a *very* efficient tool might be worth *some* extra cost. We therefore use "prioritization" in the general sense of assigning weights to each item in a dimension to reflect importance relative to the other items in that dimension.

With one exception, each dimension should be prioritized in terms of the expected importance and relevance of each item within it. The *assessment techniques* dimension is somewhat different: These techniques must also be prioritized in terms of their availability, applicability, believability (by the intended consumers of the evaluation), timeliness, and cost.

8. Apply the framework schema to evaluate and select tools. The final step is the application of the prioritized dimensions to the

candidate tools. The appropriate assessment techniques are used to evaluate the relevant metrics applied to each capability of a particular tool in each context, given the relevant application characteristics. Since the dimensions have already been pruned prior to this step, the cross-product of the evaluations performed here will be minimal (that is, as small as possible, though not necessarily small). A large number of individual evaluations may still be required, but this is unavoidable.

The thoroughness and formality with which these evaluations are made (and with which their weighted results are combined and compared to select a tool or tools) is left to the discretion of the evaluators and the consumers of the evaluation. In some cases, it may be preferable to display multidimensional, heterogeneous results by means of a graphic presentation mechanism, such as the scorecard technique described in Goeller et al. (1977) and Miser and Quade (1985).

VI. CONCLUSIONS

Many of the conclusions that emerged from this study relate to software engineering aspects of the expert system endeavor. Robustness, reliability, portability, integrability, database access, concurrent access, performance (i.e., speed and scalability of knowledge bases), and user interface all appear to be increasingly important requirements for tools, as well as eventual requirements for the expert systems that will be produced with those tools. In addition, the expert system paradigm appears to have had a significant and beneficial effect on software engineering itself.

In most cases, use of an expert system tool seems to be worth the cost. Most users believe the tools offer significant advantages over programming directly in a higher-level language (such as LISP), even though the tools do present some problems.

Tool developers and users alike are skeptical about finding practical ways of applying evaluation criteria to tools. Some of the assessment techniques suggested here have evoked enthusiasm, but it is generally believed that most of them require the creation and maintenance of an impartial database of evaluation information, which is unlikely to occur spontaneously.

Finally, we believe that the healthy evolution of the expert system field requires the infusion of new ideas from the AI research community as well as continued refinement of existing techniques by commercial tool vendors.

EXPERT SYSTEMS AND SOFTWARE ENGINEERING

One of the strongest conclusions of this study is that building an expert system is first and foremost a software engineering endeavor. This is somewhat less true when a tool is being used simply for prototyping or exploring ideas, but even in that case, issues of representation, integration with an existing development environment, debugging support, graphics capability, user interface, etc., can easily swamp issues of knowledge acquisition, problem formulation, and domain understanding. (These latter concerns can in fact also be legitimately considered part of software engineering.) For a development effort that is carried through to delivery and maintenance of an expert system, software engineering issues are even more crucial.

Many of the problems encountered (and yet to be encountered) in expert systems have already been analyzed extensively in traditional software engineering. If the expert system context places new constraints on some of these problems or sheds new light on them, it is worth reexamining the lessons learned from software engineering, but they must not be ignored.

We believe that the emergence of expert systems into the real world of software engineering will (and should) result in the recognition of expert systems as real, deliverable software that is judged by its long-term reliability and performance in end-user environments.

The Expert System Paradigm's Contribution to Software Engineering

The expert system paradigm has contributed new insights to software engineering, providing new solutions to traditional problems, as well as redefining and reinterpreting the basic software engineering agenda in a number of ways. Expert systems have encouraged a shift away from designing procedural systems to meet prior specifications and toward the explicit representation of the knowledge required to solve problems. Part of this shift can be viewed as a preference for declarative versus procedural forms, but it is really much deeper than this. It is a shift away from implicit representation of a problem (in code that solves it) toward explicit representation of the requisite knowledge, stated in the domain's own terms, along with general-purpose inference mechanisms for interpreting this knowledge to solve the given problem. This is a fundamental change that will have a profound effect on the way software engineering is performed.

Equally important is the shift away from the traditional, monotonic software engineering methodology that begins with requirements analysis and proceeds through specification, design, implementation, and testing, with little or no mechanism for backing up to rethink previous phases. In contrast, the expert system approach emphasizes rapid prototyping with continuous feedback in an iterative process that converges toward a problem solution. A prototype may evolve directly into a delivered system or it may serve as the basis for the specification of a system.[1]

Rapid prototyping does not yet play a recognized role in conventional "waterfall" models of software development (Boehm, 1981), but several software engineering technologists have expressed its potential

[1]This process was discussed by Air Force, Army, and Navy representatives at a symposium on Artificial Intelligence for Military Logistics held in Williamsburg, Virginia, March 17–19, 1987.

for increasing software effectiveness and productivity (see Ramamoorthy et al., 1984). As stated by Frederick P. Brooks, Jr.,

> Much of present-day software-acquisition procedure rests upon the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built, and install it. I think this assumption is fundamentally wrong, and that many software-acquisition problems spring from that fallacy. Hence, they cannot be fixed without fundamental revision—revision that provides for iterative development and specification of prototypes and products (Brooks, 1987).

Whether software engineering has begun to espouse the rapid prototyping approach through diffusion or through independent invention, expert system tools are among the first to support this new methodology. The separation of knowledge and inference mechanisms allows alternative designs to be implemented far more quickly than do traditional techniques, and the graphics and explanation capabilities of expert system tools allow effective feedback to both designer and domain expert. This combination facilitates quick and relatively inexpensive prototyping, refinement, and redesign. In light of these developments, we believe that DoD and industry software acquisition standards (e.g., Joint Logistics Commanders, 1982, and Department of Defense, 1987) should be reexamined to accommodate the use of rapid prototyping as a new software engineering technique.

Concerns of Tool Developers and Tool Users

Since tool vendors are in the business of developing and delivering software products, they are highly sensitized to the difficulties encountered in producing delivered software. The tools themselves have many of the characteristics of the expert systems they will be used to develop: They are large, complex, and necessarily somewhat experimental systems that must meet requirements for robustness, reliability, and integrability. Further, as the general-purpose workstation (or even the personal computer) replaces the LISP machine as the preferred delivery vehicle for expert system tools, vendors will have to provide portability and support in multi-machine markets. The tool developers we dealt with were painfully aware of the difficulty of solving these problems; they recognized that providing tools that will help users solve similar problems when developing target expert systems is a tremendous challenge that in many cases has only barely been identified.

Tool users, on the other hand, are more sensitive to issues of integration with existing hardware, software, interfaces, and database

environments. An expert system designer faced with a real application must address the issue of integration, with all its attendant constraints. With some exceptions, integration support is not yet provided by expert system tools, though it is often the criterion that determines whether an expert system project can be attempted in a given environment. This problem is especially acute for DoD and aerospace projects involving "embedded" systems that are parts of larger systems.

ADVANTAGES AND DRAWBACKS OF EXPERT SYSTEM TOOLS

The overwhelming majority of tool users are convinced that using expert system tools is well worth the expense. Most users have their personal lists of desired additions and improvements to the tools they use, yet they all seem to feel that the tools provide definite benefit and that tool vendors are generally helpful and supportive.

The new paradigm is, of course, far from perfect. Many tool users feel that the implicit, distributed control that is characteristic of most rule-based systems makes it difficult to design and understand the sequence of events that takes place during inference. If the behavior of an expert system is entirely controlled by rules, and if the order in which rules fire is indeterminate, it can be difficult to specify intended sequences of events. The use of rulesets and "control blocks," such as ROSIE's iterative rulesets and recursive procedure calls (Kipps, 1986), can help ameliorate this problem, but it will have to be solved in a more general way before full-scale expert systems can be designed with ease and confidence.

Users feel strongly that the version of a tool used for learning should be "bug-free." Even if this requires using a version that lacks crucial features, beginning users are better served by a stable, reliable piece of software than by new versions that are released prematurely.

In some cases, developers are using traditional programming languages (e.g., LISP and C) instead of expert system tools to implement their expert systems. Some of these efforts were begun when the available tools were far less powerful and far less mature than they are now. This was true only one or two years ago, but waiting for a tool to be developed would, in many cases, have involved unacceptable risk and delay. In the remaining cases, expert systems have been built without the aid of tools because of a need for either speed or flexibility.

The Need for Speed

For some developers, performance speed is an overriding concern. Projects that are amenable to algorithmic solutions may always achieve better performance by coding in efficient procedural languages, but a number of expert system tools are evolving in the direction of producing efficient code in procedural languages (such as C) and eliminating the overhead of LISP, both for themselves and for the expert systems they produce. Of course, if a given approach to solving a problem requires processes that are inherently inefficient (such as heuristic search, nondeterminism, etc.), recoding these processes in a procedural language will not necessarily improve performance. Several tools are being explored that may allow a developer to "migrate" code from heuristic to algorithmic form in certain well-constrained cases; this approach may eventually provide a partial solution to the intrinsic inefficiency of certain AI techniques.

The Need for Flexibility

Some projects have special requirements that argue for in-house tool development or direct implementation of an expert system in an underlying language. For example, a project that is concerned with research issues that require representational flexibility beyond the range of any available tool may require coding in LISP or Prolog. In addition, each tool tends to enforce a certain style of representation and control, and if a project team's mindset is at odds with all such available styles, then using a tool may be counterproductive. Special problems or environmental constraints (such as running on special hardware, integrating with special sensors or effectors, or accessing special systems or databases) might also dictate an in-house approach.

APPLYING EVALUATION CRITERIA TO TOOLS

Many of the assessment techniques we have suggested would require considerable work and cooperation on the part of tool users and vendors (for example, building and maintaining databases of published case studies or benchmark solutions). Until such work produces a critical mass of useful information, these techniques will be of limited utility. Still, they appear to represent the best that can be done at this time, and they provide a light at the end of the tunnel for decisionmakers who must choose tools. The responsibility for creating and administering appropriate databases should be undertaken by an organization that has no proprietary interest, preferably one funded either by a

government agency such as DARPA or by a consortium of tool user groups, to ensure its independence and objectivity. The raw data for evaluation of expert system tools, the knowledge about how to apply this data to a particular development effort, and the methodical translation of the data and knowledge into a specific set of applicable criteria and assessment techniques could all reside in an expert system. Such a system would advise users on how to evaluate tools in a given context. It would have the advantage of incorporating a complex body of often implicit information into an explicit form that would facilitate refinement and extension, while being easily reproducible and therefore accessible to a large community of potential tool users.

IMPLICATIONS FOR THE FUTURE

The expert system paradigm is new and evolving. Its initial transition from research to product was largely accomplished by a mass exodus of AI researchers to the commercial domain, and its continued development by commercial AI vendors seems fairly certain. However, if it is to continue to evolve, it will require the infusion of new ideas and techniques from the AI research community. The future of the field depends largely on whether the research community can continue to deliver relevant, workable ideas to developers via a "technology transfer pipeline."

If this technology transfer cannot be established, expert system technology may become stagnant and inbred. In this case, the comparison and evaluation of expert system tools will become straightforward, but the potential of the paradigm will be sharply limited. On the other hand, if a pipeline to commercial vendors is established (and we are optimistic that it will be), these tools will continue to evolve and their potential will continue to grow, but the task of comparing and evaluating them will remain a difficult one. This is by far the preferable alternative. The framework of criteria and the evaluation methodology presented here should serve as a communication vehicle for identifying and analyzing the issues surrounding expert systems and expert system tool development and as an aid in focusing expert system research and designing new tool capabilities.

The difficulty of comparing and selecting tools is largely a result of the richness of the field and the pace at which new ideas are being incorporated into tools. Our evaluation approach is not proposed as a final answer to a fixed problem, but as a strategy for dealing with a dynamic problem in a challenging research area whose impact on software engineering is only beginning to be felt.

VII. A RECOMMENDATION FOR THE FUTURE

There is clearly an emerging need for an organization to support the use and sharing of expert system applications within the DoD and the military services, using a methodology and framework such as we have defined here. We believe this to be an opportune time to create such an organization to collect, manage, and coordinate expert system tool and application information and expertise among DoD users and contractors. It would assist in using the methodology for the selection of expert system tools, increase productivity through sharing of experience and applications, and influence the development of expert system tool products.

REASONS FOR ESTABLISHING AN ORGANIZATION

Although the current tools are immature, tool builders have recently become aware of the need to "software engineer" their products. Making a current-generation tool into a software product requires freezing a product design, establishing a production process with strict control of releases and versions, and expending considerable resources. A product design obviously reflects a vendor's beliefs about what users want. If these beliefs are correct, the vendor makes a profit; if they are wrong, the vendor may go out of business, be unable to support the product, or be unable to afford an improved release of it. Since the number of expert system applications is increasing throughout the DoD, as is the number of experienced tool users (within both the DoD and its contractor community), an organization that represents these user needs could exert an influence on expert system tool products that would benefit both users and developers. In addition, since many of the DoD applications are similar to commercial applications, the organization would actually represent a wide-ranging group of users, both commercial and military.

Coordinating expert system building activities across the DoD would also increase productivity and prevent duplication of effort. Without a means for sharing experiences and results, similar applications may be reimplemented by different organizations, and different knowledge bases and tools may be poorly coordinated, making it difficult for them to work together within a system.

Some may consider it premature to form such an organization, but we disagree, since the tools are needed by users now, and users have

had insufficient experience to understand their own needs fully. The majority of users we interviewed felt they had benefited from using the tools and would use them again in the future. Most of the applications described in our users' workshop were demonstration or prototype systems, most of the users had only a few years' experience using any tool, and most had experience in using only a single tool. There seem to be relatively few fielded expert system applications (although we note that we may have a biased sample, since we lack reports on proprietary and classified systems). Users have expressed the importance of integrating expert system applications with other software and embedding them in larger systems, as well as implementing them in standard programming languages. On the whole, users and developers agree that an expert system application on a single-user LISP-based workstation is generally not acceptable for a fielded system.

Given the lack of user experience (especially with fielded systems), we believe that an organization representing DoD users' needs would be highly beneficial, even if in its initial stage it supported mainly the tools for exploratory and prototyping efforts as well as aiding future users in tool selection.

TASK DESCRIPTION FOR THE ORGANIZATION

The main responsibility of the organization would be to support the use and sharing of expert system application information within the DoD and the services to promote greater productivity, reduce duplication of effort, aid users in tool selection, and influence the future development of tool products. Ultimately, the organization might also support the sharing of domain knowledge bases among similar applications.

The organization could initially consist of a small staff that would interact with representatives from DoD agencies and the services. Those representatives would be responsible for collecting the information from their agencies and forwarding it to the organization. They would also participate in determining and reviewing the functions of the organization to ensure that it was meeting their needs.

The first tasks of the new organization would be (1) the development of questionnaires to be filled out by expert system developers and potential developers; (2) the maintenance of a database of expert system tool usage, including tool comparisons, questionnaire/interview information, benchmarks, and case studies; (3) the collection of relevant literature on expert system tools and studies; and (4) support for users in tool selection through the use of the tool evaluation methodology and database.

RESPONSIBILITY FOR THE ORGANIZATION

Possible sponsoring agencies would include a DoD agency such as DARPA, or an agency or laboratory within the Office of the Joint Chiefs of Staff (OJCS). The organization could be located within the sponsoring agency or at a National Laboratory such as Los Alamos (which is currently conducting classes in AI and expert systems); it could be part of the Software Engineering Institute at Carnegie Mellon University or a new institute at a selected university or research organization; or it could be located at a service laboratory (especially if such a laboratory were the sponsor for the organization). As an example, the Logistics Section of OJCS has set up a Logistics AI Coordinating Cell (LAICC) with the charter to be a clearinghouse for information about the use of AI in logistics for its members, the Defense Logistics Studies Information Exchange (DELSIE), and the Defense Technical Information Center (DTIC). The LAICC has representation from the four services, the Office of the Assistant Secretary of Defense for Acquisition and Logistics (OASD A&L), and OJCS. If similar organizations exist, the task could become one of consolidating them into one organization or of coordinating their activities, including the sharing of databases.

Appendix

AN ANNOTATED BIBLIOGRAPHY OF EXPERT SYSTEM TOOL EVALUATIONS

This annotated bibliography of expert system evaluation reports and studies was collected through library searches, suggestions from workshop attendees, and conversations with authors. It comprises the most current and relevant general evaluation documents. We have excluded papers that address only a particular problem, domain, machine, or methodology. Some excellent, thorough studies comparing expert system tools are also excluded because they are out-of-date or because their authors have published later and more relevant documents. We anticipate that many more evaluation reports will be published in the near future, and we expect that this bibliography will soon be outdated.

Beach, S. S., "Evaluating Expert System Tools," *The Spang Robinson Report*, Vol. 2, No. 10, October 1986, pp. 1–8.

Summary: KEE, ART, and S.1 are used to design and simulate a model train track. (Knowledge Craft was used as well; its review appears in Vol. 2, No. 11 of *The Spang Robinson Report*.) This experience, along with company interviews, customer reports, product demonstrations, and press clippings, is used to evaluate the three tools. An in-depth feature-comparison chart giving information about 18 popular tools is included.

Beach, S. S., Beach Associates, Inc. (forthcoming).

Summary: KEE, ART, Knowledge Craft, and S.1 are used to design and simulate a model train track. This commonality is used to provide an in-depth summary of each tool, and to compare each tool's knowledge representation, inference mechanisms, control techniques, and level of support available to users. The computer code is included.

Bundy, A. (ed.), *Catalogue of Artificial Intelligence Tools*, 2nd Rev. Ed., Springer-Verlag, New York, 1986.

Summary: A listing of current AI techniques and portable software (both commercial and noncommercial products) with

over 250 entries, designed to further interaction among members of the AI research community. Each piece of software or technique is briefly described, and availability information and references are given. An on-line version that is more specific to the UK can be accessed by contacting John Smith of the Rutherford-Appleton Laboratory, Chilton, Didcot, Oxon, OX11 OQX.

Culbert, C. J., *Comparison of ART and KEE*, NASA, Lyndon B. Johnson Space Center, FM7(86–8), February 11, 1986a.

Summary: ART and KEE are each used to develop two substantial expert systems, and extensive experience is used to compare the two tools. The philosophy behind each tool is explained, and the tools' main features—the requirement for LISP knowledge, the development environment, extensibility, and anticipated changes—are compared. A full evaluation of each tool is available from the Lyndon B. Johnson Space Center (KEE in FM7(86–9), and ART in FM7(86–10)).

Culbert, C. J., *Expert System Building Tools*, NASA, Lyndon B. Johnson Space Center, FM7(86–19), February 11, 1986b.

Summary: Based on experience in over half a dozen expert system development efforts, the authors describe the features necessary in any expert system tool used for NASA applications; other desirable (but not necessary) features are also noted. The report includes a feature comparison chart of 16 commonly used tools, showing the features of each. A description is given of each tool, explaining how it supports (or does not support) each feature.

"Directory of Microcomputer-Based Software for Expert Systems Work," *Expert Systems*, Vol. 2, No. 4, October 1985, pp. 222–229.

Summary: A (noncomprehensive) guide to software available for developing expert systems on personal computers. Very brief descriptions are given of each tool, along with prices.

Gevarter, W. B., *The Nature and Evaluation of Commercial Expert System Building Tools*, NASA, Ames Research Center, NASA Technical Memorandum 88331, June 1986.

Summary: A very useful general overview of what an expert system tool consists of and what it can be used for. It explains knowledge representation, inference engines, developer/user interfaces, underlying languages, and computers supported by a tool,

and it also describes the types of problems that expert systems usually attack (e.g., design, planning, etc.). A feature-comparison chart (with entries such as "what if . . . ," and pattern matching) is included, along with an attribute chart (with entries such as language of tool and knowledge base editor) for each of 20 tools, with a short synopsis of each tool.

Gilmore, J. F., and C. Howard, "Expert System Tools for Practitioners," presented at the First Australian Artificial Intelligence Conference, Melbourne, Australia, November 1986.

Summary: Discusses criteria that can be used for selecting large and small expert system tools. Thirty-seven tools are reviewed and categorized as small-scale or large-scale tools. The top 8 are then selected in both categories and a feature comparison chart is given for them. A short critique is given for each of the 37 tools. The evaluations are based on documentation, references, vendor descriptions, and discussions with sales representatives. These evaluations were originally used in the design of GEST (Generic Expert System Tool) at Georgia Tech Research Institute.

Harmon, P. (ed.), *Expert Systems Strategies*, Arlington, Mass., September 1985 through present.

Summary: This monthly newsletter aimed at managers and developers of expert systems contains reviews of new products, comparisons of tools, and other articles and announcements that might be of special interest. The most recent tool evaluations are, "M.1, NEXPERT, and Personal Consultant Plus: A Comparison of Three Mid-Sized Tools," by Brian Sawyer and Paul Harmon, April 1986; "Expert Systems Building Tools," March 1986; and "Overview of Small Expert Systems Building Tools," September 1985.

Harmon, P., and D. King, "Commercial Tools," Chapter 8 in *Expert Systems*, John Wiley and Sons, Inc., New York, 1985, pp. 92–133.

Summary: In-depth descriptions of 16 tools and a feature comparison chart (both of which are possibly outdated), along with a checklist for evaluating knowledge engineering tools which provides the reader with a takeoff point for conducting evaluations himself.

Mayer, R. J., et al., *A Characterization of Expert System Development Tools for Manufacturing Applications*, Knowledge Based Systems

Laboratory, Department of Industrial Engineering, Texas A&M University, June 10, 1986.

Summary: Geared toward expert system use in manufacturing, this paper provides a procedure for evaluating tools and for choosing the right tool for a manufacturing application. It also discusses 72 tool needs that are specific to manufacturing. A hierarchical ordering of tool features is given, along with definitions/discussions, short synopses of 8 popular tools (ART, KEE, Knowledge Craft, Loops, RuleMaster, OPS83, Prolog, and Expert), and a listing of 54 available tools, with manufacturers and prices.

Mettrey, W., Knowledge Systems Corp., for Digital Equipment Corp. (forthcoming).

Summary: In-depth feature analyses of ART, KEE, Knowledge Craft, S.1, and VAX OPS5, based on experience, manuals, and literature. Each tool is also used to implement three substantial expert systems, solving problems in selection, diagnostics, and planning.

Richer, M. H., "An Evaluation of Expert System Development Tools," *Expert Systems*, Vol. 3, No. 3, July 1986, pp. 166–183.

Summary: Discusses five main criteria for selecting an expert system tool—basic features, development environment, functionality, support, and cost. Based on available literature, user interviews, demonstrations, observations at IJCAI '85, discussions with sales representatives, and visits to companies, four tools are then evaluated in depth: S.1, KEE, Knowledge Craft, and ART. The evaluations cover the five criteria in detail and provide a comprehensive look at each tool.

Ruby, D., "Smart Systems on the PC," *PC Week*, Vol. 3, No. 9, March 3, 1986, pp. 91–94.

Summary: The approach taken is that a company interested in getting its feet wet in AI might want to start out by exploring a tool for the IBM-PC, since such tools are often simpler and less expensive than those for larger computers. Possible criteria for selecting a PC-based tool are discussed, and a large feature comparison chart is presented for 21 PC-based tools.

Walker, T. C., and R. K. Miller, "Tools for Building Expert Systems," Chapter 3 in *Expert Systems 1986*, SEAI Technical Publications, Madison, Georgia, 1986.

Summary: Short descriptions of 73 expert system tools, highlighting each tool's strengths, listing the hardware the tool will run on, and giving the tool's cost. The book is updated and published yearly. It also describes expert systems that are commercially available, proprietary programs used in-house, and projects in the prototype stage.

Wall, R. S., et al., *An Evaluation of Commercial Expert System Building Tools*, Texas Instruments Inc., Computer Science Laboratory Technical Report 85–30, November 9, 1985.

Summary: A small prototype system to monitor the readiness of the Navy's Pacific Fleet was built using each of three tools: KEE, ART, and Knowledge Craft. The experience gained therein is used to evaluate each tool's suitability for producing a much more powerful system in the same area. Although the evaluations are specific to this particular domain and the versions of the tools evaluated are somewhat out-of-date (circa 1985), insight can be gained from the techniques used for tool selection.

Waterman, D. A., "Catalog of Expert System Tools," Chap. 28 in *A Guide to Expert Systems*, Addison-Wesley, Reading, Massachusetts, 1986, pp. 339–365.

Summary: Short descriptions of 95 expert system tools, categorized as aids to system building, frame-based languages, logic-based languages, object-oriented languages, procedure-oriented languages, and rule-based languages.

REFERENCES

Anderson, R. H., and J. J. Gillogly, *Rand Intelligent Terminal Agent (RITA): Design Philosophy*, The RAND Corporation, R-1809-ARPA, February 1976.

Anderson, R. H., et al., *RITA Reference Manual*, The RAND Corporation, R-1808-ARPA, September 1977.

Barr, A., and E. A. Feigenbaum (eds.), *The Handbook of Artificial Intelligence*, Vol. I, William Kaufmann, Inc., Los Altos, California, 1981.

Beach, S. S., "Evaluating Expert System Tools," *The Spang Robinson Report*, Vol. 2, No. 10, October 1986, pp. 1-8.

Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981, pp. 35-46.

Brooks, F. P., Jr., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No. 4, April 1987, pp. 10-19.

Buchanan, B. G., et al., "Constructing an Expert System," in F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (eds.), *Building Expert Systems*, The Teknowledge Series in Knowledge Engineering, Addison-Wesley, Reading, Massachusetts, 1983, pp. 127-167.

Bundy, A. (ed.), *Catalogue of Artificial Intelligence Tools*, 2nd Rev. Ed., Springer-Verlag, New York, 1986.

Callero, M., D. A. Waterman, and J. R. Kipps, *TATR: A Prototype Expert System for Tactical Air Targeting*, The RAND Corporation, R-3096-ARPA, August 1984.

Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, Vol. 1, No. 3, Fall 1986, pp. 23-30.

Clancey, W. J., *Methodology for Building an Intelligent Tutoring System*, Department of Computer Science, Stanford University, STAN-CS-81-894, October 1981.

Clayton, B. D., *ART Programming Tutorial*, 3 Vols., Version 2.0, Inference Corporation, 1986.

Culbert, C. J., *Comparison of ART and KEE*, NASA, Lyndon B. Johnson Space Center, FM7(86-8), February 11, 1986a.

, *Expert System Building Tools*, NASA, Lyndon B. Johnson Space Center, FM7(86-19), February 11, 1986b.

Department of Defense, *Defense System Software Development* and companion documents, DOD-SDD-2167A, April 1, 1987.

Drastal, G. A., "Experiments with RULE WRITER, a Tool for Building Expert Systems," in J. P. Fry, et al. (eds.), *Proceedings of the Seventeenth Hawaii International Conference on System Sciences (HICSS)*, Vol. 2, 1984, pp. 167–173.

Fain, et al., *The ROSIE Language Reference Manual*, The RAND Corporation, N-1647-ARPA, December 1981.

Gevarter, W. B., *The Nature and Evaluation of Commercial Expert System Building Tools*, NASA, Ames Research Center, NASA Technical Memorandum 88331, June 1986.

Gilmore, J. F., and C. Howard, "Expert System Tools for Practitioners," presented at the First Australian Artificial Intelligence Conference, Melbourne, November 1986.

Goeller, B. F., et al., *Protecting an Estuary from Floods—A Policy Analysis of the Oosterschelde: Vol. I, Summary Report*, The RAND Corporation, R-2121/1-NETH, December 1977, pp. 8–13 and 126–129.

Harmon, P. (ed.), *Expert Systems Strategies*, Arlington, Massachusetts, September 1985 through present time.

Harmon, P., and D. King, "Commercial Tools," Chap. 8 in *Expert Systems*, John Wiley and Sons, Inc., New York, 1985, pp. 92–133.

Hasling, D. W., W. J. Clancey, and G. Rennels, *Strategic Explanations for a Diagnostic Consultation System*, Department of Computer Science, Stanford University, STAN-CS-83-996, November 1983.

Hayes-Roth, F., D. A. Waterman, and D. B. Lenat (eds.), *Building Expert Systems*, The Teknowledge Series in Knowledge Engineering, Addison-Wesley, Reading, Massachusetts, 1983.

Hendrix, G. G., *KLAUS: A System for Managing Information and Computational Resources*, Stanford Research Institute (SRI) International, Technical Note No. 230, October 1980.

Joint Logistics Commanders, Joint Policy Coordinating Group, Computer Software Management Subgroup, *Military Standard Defense Systems Software Development*, Department of Defense, MIL-STD-SDS, April 15, 1982.

Kipps, J. R., B. Florman, and H. A. Sowizral, *The New ROSIE Reference Manual and User's Guide*, The RAND Corporation, R-3448-DARPA/RC, December 1986.

Mayer, R. J., et al., *A Characterization of Expert System Development Tools for Manufacturing Applications*, Knowledge Based Systems Laboratory, Department of Industrial Engineering, Texas A&M University, June 10, 1986.

Miser, H. J., and E. S. Quade (eds.), *Handbook of Systems Analysis*, North-Holland, New York, 1985, pp. 89–109, 230–233.

Paul, J., D. A. Waterman, and M. A. Peterson, "SAL: An Expert System for Evaluating Asbestos Claims," *Proceedings of the First Australian Artificial Intelligence Congress*, Melbourne, 1986.

Radian Corporation, *RuleMaster-a software tool for building expert systems: Reference Manual*, January 1986a.

———, *RuleMaster-a software tool for building expert systems: Tutorial Manual*, January 1986b.

Ramamoorthy, C. V., et al., "Software Engineering: Problems and Perspectives," *IEEE Computer*, Vol. 17, No. 10, October 1984, pp. 191–209.

Richer, M. H., "An Evaluation of Expert System Development Tools," *Expert Systems*, Vol. 3, No. 3, July 1986, pp. 166–183.

Ruby, D., "Smart Systems on the PC," *PC Week*, Vol. 3, No. 9, March 3, 1986, pp. 91–94.

Shortliffe, E. H., *Computer-Based Medical Consulations: MYCIN*, Elsevier Computer Science Library, Artificial Intelligence Series, Elsevier, New York, 1976.

Sowizral, H. A., and J. R. Kipps, *ROSIE: A Programming Environment for Expert Systems*, The RAND Corporation, R-3246-ARPA, October 1985.

Van Horn, M., *Understanding Expert Systems*, Bantam Books, New York, 1986.

Walker, T. C., and R. K. Miller, "Tools for Building Expert Systems," Chap. 3 in *Expert Systems 1986*, SEAI Technical Publications, Madison, Georgia, 1986.

Wall, R. S., et al., *An Evaluation of Commercial Expert System Building Tools*, Texas Instruments, Inc., Computer Science Laboratory Technical Report 85-30, November 9, 1985.

Waterman, D. A., *A Guide to Expert Systems*, The Teknowledge Series in Knowledge Engineering, Addison-Wesley, Reading, Massachusetts, 1986a.

Waterman, D. A., and F. Hayes-Roth, *An Investigation of Tools for Building Expert Systems*, The RAND Corporation, R-2818-NSF, June 1982.

Waterman, D. A., and M. A. Peterson, *Models of Legal Decisionmaking*, The RAND Corporation, R-2717-ICJ, 1981.

Waterman, D. A., et al., *An Explanation Facility for the ROSIE Knowledge Engineering Language*, The RAND Corporation, R-3406-ARPA/RC, September 1986b.