MAPPING THE EXPERT SYSTEMS DEVELOPMENT PROCESS TO THE DEPARTMENT OF DEFENSE LIFE-CYCLE MODEL

A. F. Umar Khan
7th Communications Group (US Air Force)

ABSTRACT
This paper reports on the results of a study performed by a team from RAND Corp. and the 7th Communications Group (USAFF) for the Office of the Assistant Secretary of Defense for Production and Logistics. The study was published as: The Guide for the Management of Expert Systems Development, R-3768-P&L, and Additional Appendices, N-2797-P&L, by Iris Kameny, Umar Khan, Jody Paul, David Taylor. The study guides managers and technicians seeking expert systems (ES) solutions. As ES developments are software engineering efforts, the lessons learned building software over the past decades still apply. DoD software engineering and life-cycle policies can be applied to ES developments. Our practical methodology maps the rapid prototyping paradigm to DoD standards using the spiral model and risk containment strategies as described by Barry Boehm. Furthermore, software life-cycle management and documentation standards reflect a hardware bias which also affects conventional software developments. Other much publicized peculiarities of ES developments are not totally unique to expert systems, either. Modern conventional software developments are experiencing many of the same problems. Hence, the standards and acquisition practices of yesterday need to evolve to accommodate the nature of software development in the 1990's and beyond.

BACKGROUND
Several years ago, the Department of Defense (DoD) Logistics community earnestly began investigating expert systems technology solutions to many of its problems and it has established a Logistics Artificial Intelligence Coordination Cell centralized in the Office of the Assistant Secretary of Defense for Production and Logistics (OASD/P&L) recognized the potential for expert systems technology to proliferate throughout the DoD community; they felt that any such proliferation should be based upon sound principles learned during the successful and failed attempts to apply this technology. To that end, a team of researchers from the RAND Corporation was given a task to analyze case studies of expert systems developments accomplished within the DoD Logistics community, to review the technical and managerial literature, and to recommend a management methodology for expert systems developments. I joined the task later, representing the U.S. Air Force's 7th Communications Group (7CG).2

Objectives
Due to the relatively recent commercialization of expert systems technology, popular literature has tended to emphasize the technical issues involved (such as those related to proof of concept) to the near exclusion of the management issues. Thus, when we began our study, there were few books or journal articles which focused on key management issues (i.e., estimating the level of effort before embarking on a development project, costing new developments, calculating potential return on investment, project management, documenting the development life-cycle, integrating expert systems with the organization's existing computing base, and the like). Our study attempted to bridge this gap between technical and managerial concerns. From the outset, however, we expected that any methodology we defined should be contained in a living document -- use of our methodology in the practical solution of DoD Logistics problems would hopefully generate feedback leading to future versions which would continue to be responsive to the community's needs. Thus, we suffered no delusions that we had "the final answer."

The specific objective of our study was to provide concise information which will help managers and technicians to:

- Understand the unique issues involved in the development and operation of expert systems products as those issues relate to or diverge management;
- Assess the applicability and use of expert systems technology within their organizations;
- Identify the factors which relate to a potential for success in applying expert systems technology to a problem (this includes and the associated trade-offs vis-a-vis more conventionally understood software);
- Select the tool or tools which best suit candidate applications and the target operational environment;
- Consider expert systems in terms of larger automated information systems (AIS) architectures;
- Recognize and identify management issues relating to acquisition of expert systems tools and products, training, and maintenance; and to
- Facilitate the sharing expert systems development and operation experiences across the DoD Logistics community.

1 The specific objective of this study was to provide concise information which will help managers and technicians to:

- Understand the unique issues involved in the development and operation of expert systems products as those issues relate to or diverge management;
- Assess the applicability and use of expert systems technology within their organizations;
- Identify the factors which relate to a potential for success in applying expert systems technology to a problem (this includes and the associated trade-offs vis-a-vis more conventionally understood software);
- Select the tool or tools which best suit candidate applications and the target operational environment;
- Consider expert systems in terms of larger automated information systems (AIS) architectures;
- Recognize and identify management issues relating to acquisition of expert systems tools and products, training, and maintenance; and to
- Facilitate the sharing expert systems development and operation experiences across the DoD Logistics community.

CH2913-2/90/0000/0031$01.00 © 1990 IEEE
Our study addressed important trends observed in a survey of DoD expert systems projects and of the industry at large, including:

- Expert systems which are deployed on conventional computers, particularly mainframes and workstations, rather than on specialized platforms such as Lisp machines;
- Expert systems applications that focus on applying the technology rather than developing it;
- Applications which integrate with existing, conventional software — especially database management systems, and, to a lesser degree, spreadsheets, simulation programs, decision support systems, expert systems, and conventional programs; or,
- Applications which are prototyped on one type of hardware or language but which are delivered in another language or on another platform.

Scope

Our study primarily targeted medium to large scale expert systems development projects — particularly those efforts which require integration with other information systems technologies. Such systems inherently require more documentation and control during development than the smaller, well defined, highly constrained, stand-alone systems.

Over the past few years there has been a focus on using commercial expert systems tools to build small, well defined, stand-alone applications. This use of expert systems tools is a logical extension of the growing trend to place more computing power in the hands of the end-user. Our methodology should also be used by developers and managers of smaller development projects. It is preferable that small systems comply with the same life-cycle model as larger systems and that they adhere to similar control processes. However, obvious adjustments in management style are needed such as collapsing appropriate phases; relaxing the scope of testing, validation, and performance evaluation; reducing concern for early definition of deployment needs; and de-emphasizing long term maintenance considerations. The size and complexity of the documentation should also be reduced accordingly. In short, any reasonable management technique should be robust enough to allow for expansion or contraction to accommodate the size and complexity of the problem at hand.

Rationale

If a dozen expert systems developers are asked to describe the life-cycle most appropriate to their work, it is likely that twelve different answers will be given. Some will view the process as a single, growing prototype. Some will describe two, three, or more prototype stages. Others will argue that it is impossible to establish a single model expert systems developments vary too much depending on the application. Each will also have sound arguments to justify their preference. However, the rationale behind our suggested methodology was to evaluate expert systems developments in light of past experiences within DoD and to map those experiences to the existing DoD policy regarding automated information systems software developments. We concluded that the standards could be adapted despite the position taken by many proponents of expert systems who maintain that the paradigm it so radically different that the existing policies and software engineering practices are no longer effective. Thus, our goal was not to describe "optimum" methods for managing expert systems development by building a methodology from scratch; rather, we determined that the conventional wisdom could still be applied to this technology.

It also appears that many practitioners intentionally isolate expert systems developments from traditional MIS departments, often for political reasons. We felt that it was important that this separation not result in the removal of the traditional management "safety nets" which derive from the MIS life-cycle management and documentation standards. In short, we sought to adapt rather than to replace the existing management methodologies. We became convinced that it does not make good business sense to a priori create one set of standards for expert systems while another, familiar and workable, set of standards already exists for conventional software systems. Instead, we adopted the philosophy of helping managers and technicians live within the context of today's standards while recommending improvements for consideration in future revisions of those standards.


THE APPLICABILITY OF DOD STANDARDS

Experience building expert systems has demonstrated that:

AI technology, when properly applied, will reduce the slow software development cycle that seems to lag behind the availability of hardware by at least five years. When proper capability is available, the manner in which employees perform their jobs, how these jobs are assigned, the management decision process and the overall perception of work will be changed. [Schoen 1987]

But, one problem which has frustrated managers and technicians who might otherwise be interested in exploring expert systems solutions is the seeming unmanageability of expert systems developments. Commitment of large sums of money and of personal reputations to projects without traditional management controls is a very risky undertaking which is understandably not very appetizing for any but the most daring of managers. And, in an era of shrinking budgets, managers will be even less willing to stake their futures on anything they cannot adequately oversee, measure, and control.

A recurring theme of our work was that expert systems development is a software engineering endeavor and, as such, that it should be subject to much the same discipline as conventional software. The methodology recommended by our study is meant to be used in conjunction with accepted software development methods and standards, such as the DoD standards. Having said that, however, it is important to point out that the documentation and life-cycle standards do present a fundamental problems for expert systems development efforts.

The DoD standards implement a process-driven life-cycle model called the "waterfall" model which was described by W. W. Royce [Royce 1970] and later en-
hanced to address risk management by Barry M. Boehm [Boehm 1975] [Boehm 1976]. Expert systems development, on the other hand, is typified by an emphasis on iteration and the use of rapid prototyping. Standards which implement the waterfall model emphasize documentation which, like the milestones, must be delivered as fully complete at specified phases. This would force expert systems developers to specify their systems to an unreasonable degree of detail before they prototype. Barry Boehm now suggests that a risk-driven "spiral" model [Boehm 1988] is more appropriate for today's software development efforts than the traditional waterfall model and its variations.

This spiral model explicitly addresses prototyping, but has not matured to within DoD. Expert systems developers are, thus, faced with several significant, but surmountable, difficulties in fitting the present DoD standard life-cycle model to their projects -- particularly for contracted expert systems development. Boehm suggests that "even if an organization is not fully ready to adopt the entire spiral approach, there is one characteristic spiral model technique which can easily be adapted to any life-cycle model, and which can provide many of the benefits of the spiral approach." [Boehm 1988] That technique is a "Rice Management Plan." Our suggested methodology, therefore, addressed expert systems development in terms of existing standards but retains the spirit of Boehm's spiral model along with his suggestions regarding a risk containment strategy.

The DoD documentation standards reflect hardware requirements (e.g., memory requirements, CPU usage, etc.) rather than software requirements. This emphasis is of some value (albeit questionable) for describing conventional automated information systems because programming languages such as COBOL, Fortran, PL/1 essentially model the Von Neumann architecture at a high level they all use variables to imitate computer registers and storage cells, their control statements mimic and expand upon jump and test instructions, and assignment statements imitate fetching, storing, and arithmetic functions of the hardware [New Science 1989]. After all, these languages evolved a single step from assembly languages (languages which made programming easier by allowing mnemonic names for physical hardware and hardware related functions). Expert system development languages and tools, on the other hand, attempt to model human performance and thought processes to some degree. They tend to require more computing resources than traditional languages because of this extra level of abstraction which must still ultimately be translated into machine functions and addresses. Given this situation, standards which reflect hardware rather than software requirements effectively force the technical writer to document the translation of the expert systems application as it affects the hardware, not as it appears to either the end user or the programmer. Hence, such standards are harder to enforce on expert system developers because their documentation is necessarily hard to generate and hard to use. As noted elsewhere in this paper, though, these problems are not unique to expert systems; the proliferation of fourth generation languages, database management systems, decision support systems, spreadsheets, and the like make documentation "by the book" equally difficult for more conventionally understood applications as well.

A MANAGEMENT OVERVIEW OF ES DEVELOPMENT

In a recent study prepared for the U.S. Air Force's Rome Air Development Center [Bardwil 1987], the following expert systems software development problems were found to be most commonly mentioned in the public literature:

- There is no clear guidance on how to produce a well-engineered product using the rapid prototyping method.
- There is little known about the management of deliverables which are evolutionary in nature.
- There are a limited number of software engineers trained in expert systems development.
- There is a need for specialists in the field of AI.
- There is a problem with expert systems developments because, with rapid prototyping, the development system is always in a state of flux.
- There exists a lack of knowledge regarding testing of an expert system.
- There is often a voluminous amount of information in the definition of an Expert System.

Many of these problems have come about because of the shifting software engineering paradigm from rigid, linear-sequenced development (e.g., strict adherence to a waterfall model) to one that encourages iterative development. This departure from previously accepted theory, gradually being accepted for all software development, is a fundamental aspect of expert systems technology. One of the attractive features of the expert systems development paradigm is its exploitation of rapid prototyping. Confusion over the differences between the historical and iterative nature of the expert systems development process makes it impossible to provide management with the familiar, narrowly defined milestones and and technological merit.

To insure the long term credibility and viability of this potentially valuable technology, management must receive the usual assurances of organizational support (such as funding) as they are accustomed to in any software development effort. Hence, managers and technicians need to understand how expert systems developments relate to traditional software development efforts and how they differ.

The methodology utilized in developing AI systems differs somewhat from that used in developing other computer-based products. Project leaders whose experience has taught them how to manage conventional software development programs successfully find that they must modify or augment their management techniques. Certainly some familiar project management controls are still appropriate: schedule and cost milestones, design reviews, design specifications, test planning, etc. There are, however, some phases of an AI development project which are quite different. [Schoen 1987]

Expert systems development projects are software engineering efforts and, as such, the lessons learned building software over the past decades still apply. Insistence upon treating expert systems development projects as a specie outside of controls will contribute
to the difficulty of carrying expert systems projects to full implementation and deployment.

Organizations, such as DoD, generally standardize and regulate the development of automated systems. Strong arguments have been put forward to show how standards do not accommodate the iterative, prototyping nature of expert systems development. Within DoD, prototyping has been an acceptable, albeit underutilized, development approach for years, but the DoD standards have historically been silent on this development without precluding it.

[DoD Standard 7935.1 Draft] is probably the first DoD standard to explicitly address prototyping; but, adapting even that standard to prototyping requires some imagination on the part of the expert systems developer. There are three significant areas where the traditional software engineering methods used within DoD fall short when applied to expert systems development projects: interpretation of the expert systems life-cycle model, documentation, and design emphasis. Each of these is described below.

**THE EXPERT SYSTEM LIFE-CYCLE MODEL**

A mapping of the expert system development process to the standard life-cycle model can be accomplished in a relatively straight-forward manner. A recommended mapping of the expert systems development life-cycle to the phases prescribed in [DoD Directive 7920.1] is offered in Figure 1. Using such a mapping, existing standards for software development can be applied to an expert systems development project. Management controls must be used so that the proper overall software that proper testing of each prototype stage is planned and executed. [DoD Directive 7920.1] However, understanding just what management controls are appropriate for each stage of expert systems software development is incomplete. Over the past four or five years, the dominance of the market by platform-specific commercial shells aimed at the development of stand-alone systems has provided greater experience with the Initiation and Prototyping phases than with Deployment and Post Deployment.

![Expert System Development Life-Cycle Mapped to DoD Directive 7920.1](image)

The following paragraphs summarize the six phases of the software development life-cycle (detailed descriptions, including the key risk factors for management and technicians are discussed in The Guide for the Management of Expert Systems Development, R-3766-P&A).

**The Initiation Phase**

The Initiation Phase is concerned with deciding what is wanted. This phase deals with determining what the problem is, whether it is feasible to apply expert systems technology to the problem, and how to begin exploring the problem. The culmination of this phase is a Milestone 0 review which, if successful, brings the project to the next phase.

**The Concept Phase**

The Concept Phase involves further exploration of the problem and deciding how to best solve it. Several focused prototypes, each testing a highly specific area of risk, may demonstrate how the general problem solution will work, how parts of the problem solution that strain expert systems capabilities might be handled, how the user interface may be structured, how integration with other automated information systems or special devices will be handled, and how security concerns may be handled. Exploring the problem through rapid prototyping requires a team effort of developers, users and ADP staff, and the availability of in-house tools which are rich in functionality and flexibility. This prototyping phase will, in all likelihood, include conventional ADP issues as much as or perhaps even more than expert systems issues. It is a function of where the hard problems and the chief risks are thought to be. In complex, integrated systems, the expert systems issues may be very well understood while the software interfaces between different tools may be viewed as the real problem. In that case, it would make sense to concentrate more on the integration issues during this phase. The idea is, after all, to reduce the risk of proceeding further with the development, not just to demonstrate that expert systems offer the solution to a problem.

Concept Prototyping is the first of four prototype stages in the development of an expert systems using our risk management approach. All stages require a team support effort--particularly the Knowledge Engineer and development staff working closely with the domain expert(s) and end users. The Concept Phase culminates with a Milestone 1 review which, if successful, brings the project to the next phase.

**The Definition/Design Phase**

The Definition/Design Phase is concerned with the development of the final design for the problem solution. This design is embodied in a working prototype and associated documentation. The phase is broken down into two stages, corresponding to the two prototypes used for design development: the Demonstration Prototype and the Testbed Prototype.

The Demonstration Prototype focuses on rapid design and prototyping of the problem solution in a stand-alone mode by the Knowledge Engineer through extensive interaction with the expert(s) and to a lesser degree with end users. The expert systems tools used in this stage is primarily should offer extensive support for debugging, graphics, knowledge acquisition, multiple modes of reasoning and knowledge representation. The major goals of this stage are to correctly demonstrate the problem solution and an adequate user interface. The Demonstration Prototype is operational execution. The interfaces with other automated information systems and special devices should at least
be stubbed and the data exchanges simulated. If security concerns are a factor, proof-of-concept should be demonstrated by a walk through as a minimum.

The Testbed Prototype is a redesign and reimplementation of the functionality defined and demonstrated in the Demonstration Prototype plus possibly additional enhancements. It may be implemented using the expert systems tool that will be used in the deployed system (or the developer's version of the tool), or coded directly in a DoD acceptable programming language. Correctness is required of the Testbed Prototype and formal test and evaluation (T&E) should validate that it meets its requirements as described in the Functional Description. Performance issues may begin to be addressed at this stage or they may be delayed until the Operational Prototype. Again, integration interfaces will be implemented but may be stubbed or simulated. Security aspects of the workstation environment should be implemented and tested. This stage culminates with a Milestone 2 review.

The Development Phase

The Development Phase produces the Operational Prototype, whose objective is to integrate the Testbed Prototype into an example operational environment using the actual deployed tool, automated information systems, and special devices. Correctness and performance should be validated as should requirements for integration and security. At the completion of the Operational Prototype, the expert system is ready for deployment. The final Operational Prototype is the baseline system. The Development Phase culminates in the Milestone 3 review which, if successful, carries the project into the Deployment Phase.

The Deployment Phase

In the Deployment Phase, the fully integrated system is delivered to the field. This phase culminates in a Milestone 4 review and the system is turned over to the support organization that will maintain it.

The Post-Deployment Phase

Post-Deployment concerns maintenance of the system, evaluation of the use of the system to determine lessons learned, and planning enhancements to be implemented in system upgrades.

ES LIFE-CYCLE MANAGEMENT CONCERNS

In keeping with the spiral model for software development proposed by Barry Boehm [Boehm 1988], the management issues addressed in our study focused on the identification and containment of risk. The "GO/NO GO" decision at each phase transition in the development life-cycle should be based, to a large extent, on the ability to identify and ameliorate the risks involved such that, as the project evolves, the user's mission needs. Boehm identifies the following elements of a Risk Management Plan [Boehm 1988]:

- Identify the top ten risk items for the project.
- Prepare and present a plan for resolving each risk item.
- Update the "top ten" list of risk items, the plan, and the results monthly.
- Highlight each risk-item's status on a monthly basis during project reviews, comparing it with the ranking and status from the previous month.
- Initiate appropriate corrective actions as indicated in the plan.

DOCUMENTATION

Software documentation standards are generally narrow interpretations of more flexible models. While the underlying model might lend itself to certain types of functional documentation, the standards, not the model, actually establish the number, format, and content for those documents; they regulate when those documents are prepared and, they may also stipulate when documents are to be baseline. Such documents generally exist for:

- Provide information required to support management decisions at key transition points in the development life-cycle; and,
- Assist the maintainers of software.

For example, [DoD Standard 7935.1] describes 13 major documents using the two-letter abbreviations shown in Table 1.

<p>| Table 1 |</p>
<table>
<thead>
<tr>
<th>DOD DOCUMENT ABBREVIATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD</td>
</tr>
<tr>
<td>SS</td>
</tr>
<tr>
<td>US</td>
</tr>
<tr>
<td>OM</td>
</tr>
<tr>
<td>DS</td>
</tr>
<tr>
<td>UM</td>
</tr>
<tr>
<td>EM</td>
</tr>
<tr>
<td>MM</td>
</tr>
<tr>
<td>PT</td>
</tr>
<tr>
<td>RT</td>
</tr>
<tr>
<td>JP</td>
</tr>
<tr>
<td>RD</td>
</tr>
<tr>
<td>PS</td>
</tr>
</tbody>
</table>

Figure 2 shows those standard DoD documents prepared and used across several phases of development. These figures are extracted from [DoD Standard 7935.1] and [DoD Standard 7935.1 Draft]; they reflect a changing attitude as to when the importance of certain documents (e.g., the Functional Description, System-Subsystem Specification, Test Plan, and Implementation Procedures) begins. The supporting narrative in [DoD Standard 7935.1 Draft] avoids a commitment to any delineation of the importance of certain documents as to where in the development process document preparation begins and use begins. Hence, it is acceptable within the framework of this standard, for a document such as the Functional Description or System-Subsystem Specification to be
### Table: Definition, Development, Deployment, and Post-Deployment Phases

<table>
<thead>
<tr>
<th>Initiation</th>
<th>Concept</th>
<th>Definition and Design</th>
<th>Development</th>
<th>Deployment and Post-Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Definition</strong></td>
<td><strong>Design</strong></td>
<td><strong>Development and Integration</strong></td>
</tr>
<tr>
<td>FD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2. Standard DoD Documents Mapped Across Multiple Development Phases**

**Document Preparation and Use per DoD Standard 7935.1, April 24, 1984**

**Document Preparation and Use per DRAFT DoD Standard 7935.1, January 15, 1988**

---

updated throughout the life of a project without ever forcing it to be baselined.

**Recognizing Weaknesses Due to Emphasis**

Most software documentation standards emphasize early 1970s hardware perspective. This complicates the programmer's job. For expert systems developments there are two particular problems caused by documentation which is hardware-driven: (1) documenting functionality contributed from support tools, and (2) dealing with the categorical split of documentation for data and procedures.

**Impact of Commercial Tools on Development**

Commercial knowledge-based-system building tools differ from each other in how they represent knowledge in the knowledge base (e.g., rules, frames); the inference strategies they support (e.g., forward chaining, backward chaining); the extent and type of interfaces they provide for the developer (e.g., program libraries, reusable source code) and the end user (e.g., windows, graphical); and their support for integration with external hardware and software (e.g., specific application layer protocols). The better commercial tools help programmers concentrate details. To paraphrase Peter Szolovits [1987], the principle objective of consideration. As in the case of evaluating third generation languages, there is probably no "best" language overall, only a variety of well and design trade-offs which affect the cost and performance of those tools. Our study, building largely on previous RAND research [Rothenberg 1987], discusses the technical processes of tool selection in detail.
Today's tools impact the development of expert systems in two ways: by contributing functionality to the expert system itself, and by offering developers an option to use different tools for different phases of the lifecycle.

Documenting functionality acquired from support software tools (e.g., shells and the like) is difficult. Some expert systems development tools allow for wholesale cannibalizing of code. Some tools provide libraries of routines which can be used by expert systems developers to provide specific common functionalities. All tools lend themselves to certain capabilities they provide. Thus, the development tool actually contributes functionality to the final expert system design and should be considered as part of the system specification. [DoD Standard 7935.1 Draft] addresses this contribution by providing pointers to the nature of the functionality borrowed from a support software tool.

The diversity in capability, performance, and features, coupled with hardware and software compatibility concerns, and system performance during the life cycle. Figure 3, adapted from [Rothenberg 1987], shows the relative importance of some of the principle factors in expert systems-building tool selection across the development phases. Figure 3 can be looked at as potentially describing the use of one tool during one phase (e.g., Concept) and another tool for a different phase (e.g., Deployment). The increased productivity of some commercial expert systems accept the transition costs of moving to another tool for deployment.

Several expert systems projects under development for the Government have experienced unplanned changes in tools. Development began using one of the expert systems shells but, by the time the systems entered their later prototype stages, more than 50% of the code had to be rewritten in Lisp or C due to performance constraints. Prospector, a famous expert system developed for the U.S. Geological survey, was first developed as a rule-based expert system in InterLisp on a DEC KL-10 computer in the mid-seventies, then translated into MacLisp, then transferred to a VAX 11/750; it is now being translated back into InterLisp and rehosted on a Xerox 1108 Artificial Intelligence Workstation and redesigned to be a frame-based expert system. [McCannon 1990]. Other expert systems have experienced similar transitions over time due to changing requirements and the growth of new technologies.

Expert systems tool selection is a difficult process with many unique capabilities and features available from commercial off-the-shelf software. There is much unfamiliar ground to be explored since there is a lack of wide-spread experience with the tools and most organizations lack exposure to a range of successful expert systems development projects. There is also an element of inertia since developers may be uncomfortable with the idea of changing tools during the different expert systems development phases. Conventional software development projects have typically used the same tool(s) throughout development; although, this too is changing with the increased popularity of fourth generation languages and computer-aided software engineering tools.

(1) Tools for Development. Some tools enhance programmer productivity for large/complex applications. They may offer such productivity gains that the decision to use one tool for development and another for delivery may be justifiable. As illustrated in Figure 4 a development tool would need to be extremely flexible with very little emphasis on operational efficiency.

(2) Tools for Deployment. As integration with existing computing bases and the need for real-time systems rises, the appeal of optimizing an expert systems component by rewriting it in a higher order language, or using procedural (higher order language) code with coding from expert systems tools, can only be expected to increase. Some tools currently produce procedural code (e.g., FORTRAN, C) directly from the knowledge engineering language. Some tools may be more strongly typed and structured than others, or they may offer support for compilation or compilation of their knowledge-bases. Such tools are usually themselves well to exploratory programming. As illustrated in Figure 4, a deployment tool would need to be very efficient with a dramatic lessening of emphasis on flexibility.

Documentation which describes data in one place and procedures in another is inappropriate for documenting certain expert systems components. Object-oriented programming, for example, is a non-procedural method of programming which evolved out of Simulation and Modeling. In object-oriented programming, both data and procedures are bound together into elements called objects. In response to messages passed to them, objects may take action, send messages to other objects, or are acted upon. Internally, objects are both data and procedures. Standard documentation within DoD typically requires data and procedures to be described in separate formats and in separate documents. This separation of data and program specifications cannot be used to describe how an object functions in object-oriented programs.

OVERCOMING DOCUMENTATION WEAKNESSES

As stated earlier, software documentation serves either as an input to management or as an aid to programmer/analysts charged with maintaining software written by others. When applied to expert systems projects, these two document missions call for two different types of approaches: new forms of documentation; and, new ways of preparing the existing standard documents.

New Forms of Documentation

The previously quoted study prepared for the Rome Air Development Center [Bardawi 1987] identified several entirely new document types and formats specifically suited to the acquisition of AI software. While extensive redefinition and restructuring of the standard documentation may be justified, to do so for the exclusive case of expert systems development implies a uniqueness of this technology which probably does not exist. There is, however, one addition to the list of standard DoD documents which might benefit many types of software development: a Progress Notebook. The purpose of the Progress Notebook should be to record those activities which might otherwise never be permanently documented, those which linger only in the memories of the development team members, but knowledge of which would be of immense value to future maintainers of the resultant expert system.
In principle, a Progress Notebook would be a living document, an historic trace of all alternatives and decisions, open issues, blind alleys pursued, and the like. It would fill in the gaps left by the standard documents and documents. The Progress Notebook would contain a wide variety of information about the resulting expert system.

For ease of reference and to avoid duplication, Table 2 lists the standard documents which are recommended for inclusion in the Progress Notebook.

Table 2

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Document Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD</td>
<td>Functional Description</td>
</tr>
<tr>
<td>SS</td>
<td>System/Subsystem Specification</td>
</tr>
<tr>
<td>US</td>
<td>Software Unit Specification</td>
</tr>
<tr>
<td>DS</td>
<td>Database Specification</td>
</tr>
<tr>
<td>PT</td>
<td>Test Plan</td>
</tr>
<tr>
<td>RT</td>
<td>Test Analysis Report</td>
</tr>
<tr>
<td>IP</td>
<td>Implementation Procedures</td>
</tr>
</tbody>
</table>

In addition to the standard documents, the Progress Notebook should also include the following useful information:

- List of people involved in the effort: people in the user, developer, MIS, training and maintenance organizations as well as all people consulted about the project.
- Notes from meetings held: dates, attendees, subjects covered, action items, and critical information related to issues discussed, decisions made and rationale, project organization decisions, discussions and evaluation of risk factors.
- Transcripts from knowledge engineering sessions and notes on the representation strategies considered and accepted, explaining the mappings from the expert's domain knowledge to the form implemented in the software.
- Descriptions of briefings given: dates, purposes, attendees, copies of material presented, results of briefings.
- Documentation of information collected during the technical tasks (descriptions of the problem, problem environment, written data/knowledge sources, user community, management attitudes about the problem and expert systems technology and all identified risks in these areas).
- Documentation of the risk analysis, prioritization of risks and recommendations for risk resolution; feasibility analysis and recommendations; notes or references to similar expert systems products or development efforts and the rough cost estimate.
- Project reviews: dates, attendees, copies of materials presented, result of review including all action items and dates for follow-up (in particular the milestone reviews).
- "Problem" reports and recommendations as a result of the prototype development and Test and Evaluation.

The Progress Notebook should be structured to facilitate broad common access to its components (e.g., contractor reviews or internal product evaluations). In the best of all possible worlds, this document would be on-line, preferably using some hypertext tool. But in the real world, a three-ring binder may have to suffice. Lacking the facilities to maintain the Progress Notebook on-line, the project manager should keep it in a well-organized binder with plenty of room for insertion and expansion.

Figure 3  Relative Importance of Tool Features Across Development Phases (from [Rothenberg 1987])
New Ways of Preparing Old Documentation
As discussed earlier, premature insistence upon fully enumerated documents, such as the functional and system specifications, can be detrimental to an expert systems development effort. This can be overcome if expert systems incrementally throughout the development life-cycle. Updated at the transition points between development phases and prototypes, each design document should describe the system through its current stage (to some reasonable degree of thoroughness) and only refer generally to stages further along in the life-cycle. This approach is consistent with [DoD Standard 7935.1], as described previously.

ACQUISITION STRATEGIES FOR ES DEVELOPMENTS
Contracted software acquisition is, by nature, less flexible than in-house development.

Internal software developments have a great deal of flexibility and freedom to accommodate stage-by-stage commitments, to defer commitments to specific options, or to accommodate such practices as prototyping, evolutionary development, or design to cost. The world of contract control. [Boehm 1988]

A review of case studies revealed several poignant examples of an almost crippling impact on expert systems development efforts from contracts which demanded extensive, detailed documents at points which were out of sync with the expert systems life-cycle model.

[Bardawil 1987] recommended modifications to the DoD procurement processes in order to specifically accommodate the acquisition of AI software. Barry Boehm points to recent progress in "establishing more flexible contract mechanisms, such as the use of competitive front-end contracts for concept definition or prototype fly-offs, the use of level-of-effort and award-fee contracts for evolutionary development, and the use of design-to-cost contracts." [Boehm 1988]

However, as Boehm also points out, procedures for using these new contracting mechanisms "need to be worked out to the point that acquisition managers feel fully comfortable using them." [Boehm 1988]

SUMMARY
Many people believe that the standard life-cycle models and software engineering techniques do not apply to expert systems developments. So, managers typically have not received the familiar assurances of manageability, i.e., metrics of a project's credibility based upon business and technological merit. However, traditional software engineering methods and software development standards used within DoD do relate well to expert systems projects. Where mismatches do exist, they are usually due to narrow interpretations of the standard life-cycle model or they are due to impacts modern software development efforts.

Two models of the software development life-cycle were discussed: the waterfall and the spiral. Practically all DoD and industrial software waterfall model. The spiral model, however, is risk-driven and is appropriate for use in building expert systems applications. A mapping of the DoD life-cycle model to expert systems developments was offered as a way in which the DoD software development standards could be applied to an expert systems project and it is followed throughout this guide.

MIS organizations typically prescribe the early baselining of standard deliverables before design can begin. When applied to expert systems developments, this approach quickly becomes obsolete. Our methodology shows one way in which this can be overcome: by allowing expert systems developers to prepare and deliver the standard documents incrementally. This approach was shown to be acceptable within the context of the DoD standards but is difficult to apply when contracting for expert systems software development support and services.

CONCLUSIONS
There has been a rise in the popularity of prototyping among MIS professionals; more and more vendors are providing programmer productivity enhancement aids such as computer-aided software engineering tools and fourth generation languages. So, nearly every area of perceived uniqueness in expert systems development has a counterpart in traditional software endeavors. Both the traditional MIS community and the expert systems community need to cooperate to ensure that software engineering disciplines and acquisition strategies keep pace with the changing nature of software development:

- Where necessary, software professionals developing expert systems should re-examine the traditional wisdom about life-cycle management and documentation which was appropriate for the 1980's rather than seek separate solutions for expert systems.
- Policy makers responsible for the development of standards for software should continue to strive for up-to-date understanding and accommodating today's advanced software tools and requirements (both expert system and otherwise).
- Contracting officers should work closely with project officers who are responsible for the acquisition of expert systems development services to ensure that the proper contracting vehicle is used (i.e., one which provides for traditional sorts of deliverables within a flexible framework supportive of the rapid prototyping paradigm.

REFERENCES


NOTES

1. While this sort of oversight does not exist in DoD today, there are voices in the wilderness advocating greater centralization. Under DoD contract MDA003 85-C-0139, Logistics Management Institute recommended three organizational actions for OASDP & L to implement: the creation of a DoD Logistics AI Policy Council to be chaired by the Assistant Secretary of Defense (Production & Logistics); the establishment of a DoD Logistics Artificial Intelligence Program Office to assist the Policy Council; and, the establishment of DoD wide Logistics Artificial Intelligence Technology Transition Center "as a permanent group to provide informational, technical, and managerial support services." [1-MI 1989]

2. The 7th Communications Group is an Air Force organization providing a wide range of communications and information systems support to the Office of the Secretary of Defense (OSD), the Office of the Secretary of the Air Force, and Headquarters U.S. Air Force.

3. Our references to size were not intended to imply the use of any metric such as numbers of rules or frames; rather, size was meant as a subjective estimate of complexity, depth of reasoning, and degree of integration with the organization's existing computing base.

4. DoD Standards [2167L, 2167A], [7935.1] and [7935.1 Draft], DoD Directives [19892] 1, [5000.29], and [5000.29 Draft], and (DoD Instruction 7920.2).