NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TESTING, VALIDATION, AND VERIFICATION OF AN EXPERT SYSTEM ADVISOR FOR AIRCRAFT MAINTENANCE SCHEDULING (ESAAMS)

by

Christian W. Andrieu

March 1991

Thesis Advisor: Martin J. McCaffrey

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): Code AS	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) TESTING, VALIDATION, AND VERIFICATION OF AN EXPERT SYSTEM ADVISOR FOR AIRCRAFT MAINTENANCE SCHEDULING (ESAAMS)

12 PERSONAL AUTHOR(S): Andrieu, Christian W.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1991 March	95

16 SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Expert Systems, Validation Expert Systems, Testing Expert Systems, Aviation Maintenance

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

Aircraft maintenance control operates in a dynamic, high intensity environment. Maintenance work priorities are made several times daily under extremely demanding and time sensitive conditions. The person responsible for scheduling aircraft, usually the Maintenance Master Chief, draws upon years of experience when assigning priorities for both scheduled and unscheduled maintenance. An Expert System Advisor for Aircraft Maintenance Scheduling (ESAAMS) is being implemented at the Naval Postgraduate School. This thesis examines what should be included within an expert system test plan and proposes a prototype test plan for ESAAMS. Development of ESAAMS will provide valuable insight for incorporation of a leading edge technology into today's complex military.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL: Martin J. McCaffrey	22b TELEPHONE (Include Area Code): (408) 646-2488	22c OFFICE SYMBOL: AS/Mf

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

S/N 0102-LF-014-6603 Unclassified

#19 (Continued)

The potential improvement in operational readiness, consistent decision making, and ability to replicate an expert's decision making process for scheduling aircraft maintenance makes implementing ESAAMS a worthwhile venture.

Approved for public release; distribution is unlimited.

Testing, Validation, and Verification of an Expert System Advisor For Aircraft Maintenance Scheduling (ESAAMS)

by

Christian W. Andrieu
Lieutenant Commander, United States Navy
B.S., University of New Orleans, 1977

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1991

ABSTRACT

Aircraft maintenance control operates in a dynamic, high intensity environment. Maintenance work priorities are made several times daily under extremely demanding and time sensitive conditions. The person responsible for scheduling aircraft, usually the Maintenance Master Chief, draws upon years of experience when assigning priorities for both scheduled and unscheduled maintenance. An Expert System Advisor for Aircraft Maintenance Scheduling (ESAAMS) is being implemented at the Naval Postgraduate School. This thesis examines what should be included within an expert system test plan and proposes a prototype test plan for ESAAMS. Development of ESAAMS will provide valuable insight for incorporation of a leading edge technology into today's complex military. The potential improvement in operational readiness, consistent decision making, and ability to replicate an expert's decision making process for scheduling aircraft maintenance makes implementing ESAAMS a worthwhile venture.

TABLE OF CONTENTS

I. INTRODUCTION

Expert systems are prevalent throughout the commercial sector of industry. They serve many useful purposes such as aiding in the troubleshooting of sophisticated electronic systems, and diagnosing medical ailments. Many expert systems have resounding success stories, saving corporations which have employed them millions of dollars.

Expert system development has not been as actively pursued in the Department of Defense. The lack of funding has often been cited as the primary reason. Students at the Naval Postgraduate School are implementing an expert system advisor for aircraft maintenance scheduling (ESAAMS).

This thesis will examine what should be included within an expert system test plan and proposes a prototype test plan for ESAAMS. Methods of validating and verifying expert systems will be discussed extensively.

This chapter discusses the background, purpose, scope, and limitations of this study. Software used in the thesis is briefly discussed, followed by a chapter by chapter synopsis of the study.

A. BACKGROUND

Aircraft maintenance control operates in a dynamic, high intensity environment. Maintenance work priorities are made several times daily under extremely demanding and time

sensitive conditions. ESAAMS is being developed to assist maintenance control personnel in the prioritization of repairs for mission critical aircraft.

A critical aspect of any software life cycle is the development of a testing and evaluation plan. Proper testing will provide developers with the limitations of the system. System evaluations should be conceived, and a comprehensive test plan written in the earliest phases of design. Planning tests early in the life cycle forces developers to define specific objectives the expert system is expected to accomplish.

For an expert system part of the testing process is the actual validation and verification of the knowledge base. Simply running test programs and comparing results to those of the expert is not enough. Qualitative and quantitative measures must be developed to accurately measure the capabilities of the system.

B. PURPOSE

This thesis develops a prototype testing plan to assist in the development of an aircraft maintenance scheduling system advisor. Specific testing, validation and verification (V & V) measures will be examined for expert systems in general, then applied towards development of a test plan for ESAAMS. Limitations of the expertise to be developed will be explored, as well as the required level of response and accuracy demanded by the user. Without meeting the user's acceptance

criteria, support for the development of ESAAMS will likely disappear rapidly.

C. ESAAMS

ESAAMS is a software program currently under development by students at the Naval Postgraduate School. Its purpose is to assist maintenance control personnel within a Navy squadron in scheduling aircraft for planned and unscheduled maintenance. Using the expertise of maintenance control experts, advisory decisions on which aircraft to schedule for maintenance during a given timeframe will improve squadron readiness, while assisting maintenance control in making more consistent scheduling decisions.

D. SOFTWARE DEVELOPMENT TOOLS

Nexpert Object, Version 1.1 (Neuron Data Inc., Palo Alto, CA) has been chosen as the expert system shell for development of ESAAMS. Availability, vendor support, and the shell's ability to represent knowledge in various dimensions were the primary motivators in selecting NEXPERT OBJECT. Several testing and debugging tools are available within the software, and will be explored later in the thesis.

E. SCOPE AND LIMITATIONS

Aircraft maintenance scheduling using expert systems was determined to be feasible by McCaffrey [Ref. 1]. During the initial stages of research on this thesis, a fighter/attack (VFA) squadron from Lemoore NAS, CA., was identified for

participation in the development of ESAAMS. The maintenance control master chief was interviewed on one occasion for the purpose of gathering knowledge and his 'rules of thumb' for decision making. Further interviews and initial prototype development were planned accordingly. Once prototype development was completed, testing of ESAAMS was to follow. Due to operational commitments, the VFA squadron unexpectedly deployed before further development of ESAAMS was possible.

Continued progress on ESAAMS used the aircraft maintenance experience of the author. A proposed test plan developed for a generic aircraft squadron has been incorporated within this study. No actual testing has been completed due to the development delay of the prototype expert system caused by the nonavailability of the squadron.

An exhaustive literature review of testing, validation, and verification of expert systems indicates that little has been documented in this area. Quantitative measures have not been formally developed to assist the system designer in validating and verifying an expert system. However, an increase in research on this topic is beginning to appear in the literature. Several software packages are in early phases of development for testing of expert system knowledge bases and will be discussed later in the study.

F. ORGANIZATION OF THE STUDY

Chapter II focuses on testing and evaluation within the life cycle of an expert system. Phases of the life cycle are described, with prototypes mapped into the life cycle.

Chapter III details the Validation and Verification (V & V) process and looks into formal testing procedures. An indepth study of qualitative and quantitative means for measuring expert system validity is examined.

Chapter IV describes software debugging techniques for expert system knowledge bases. Testing and debugging tools within NEXPERT OBJECT are described. Finally, specific software available for testing expert systems is examined.

Chapter V contains a proposed testing plan to be incorporated within the life cycle of ESAAMS.

Chapter VI summarizes the research and makes several recommendations based upon the research findings.

II. EXPERT SYSTEMS LIFE CYCLE

Expert systems have been one of the most emphasized areas in Artificial Intelligence, with research producing scores of applications for industry and some initial applications for the Department of Defense (hence, development of ESAAMS for military aircraft scheduling). Feigenbaum defines an expert system as: "an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution." [Ref. 2:p. 1] It is the very nature of the expert system, that of representing human knowledge, as well as dealing with uncertainties that makes expert systems different from conventional software, and inherently more difficult to test, verify and validate.

An expert system differs from more conventional software programs in several important respects. Duda observes that in an expert system," . . . there is a clear separation of general knowledge about the problem (the rules forming a knowledge base) and methods for applying the general knowledge to the problem (the rule interpreter)." [Ref. 2:p. 6] In a conventional computer program, knowledge pertinent to the problem and methods for utilizing the knowledge are all intermixed, making it difficult to change the program. In an expert system, " . . . the program itself is only an

interpreter (or general reasoning mechanism) and (ideally) the system can be changed by simply adding or subtracting rules in the knowledge base." [Ref. 2:p. 6]

Although easy enough to do (add or subtract rules in the knowledge base), consequences may be disastrous. Reasoning throughout the program is altered when a single rule is fired incorrectly, or not fired (or is missing altogether). Thorough testing is required to detect adverse changes to the system, or in the case of correct knowledge, prove the system correct.

The goal of the software life cycle model is to ensure development of a structurally sound, viable, and usable software package within time and budgetary constraints. This chapter discusses the spiral model used for expert system software development and testing within the expert system life cycle.

A. SPIRAL MODEL

Validation methods for software are difficult, time consuming, and expensive. Correctness is not a sure thing, as evidenced by errors found after software is released, requiring the necessity of continued maintenance. Traditional software validation is represented by the waterfall model, [Ref. 3], where the life cycle evolves from the requirements stage and leads into specification, design, and coding. This methodology is not oriented towards the incremental development method typical of knowledge based systems.

The spiral model, Figure 1, adapted from Boehm, [Ref. 4] most closely approximates an appropriate developmental approach for a new technology addressing ill defined problems, where heuristics have a dominant role.

The radial dimension represents cumulative cost and the angular dimensions represent progress made in completing each cycle of the spiral. Starting in the innermost spiral, and radiating outward in a clockwise fashion, each new cycle begins by defining risk analysis, development of a prototype, testing, verification, and planning for the next phases.

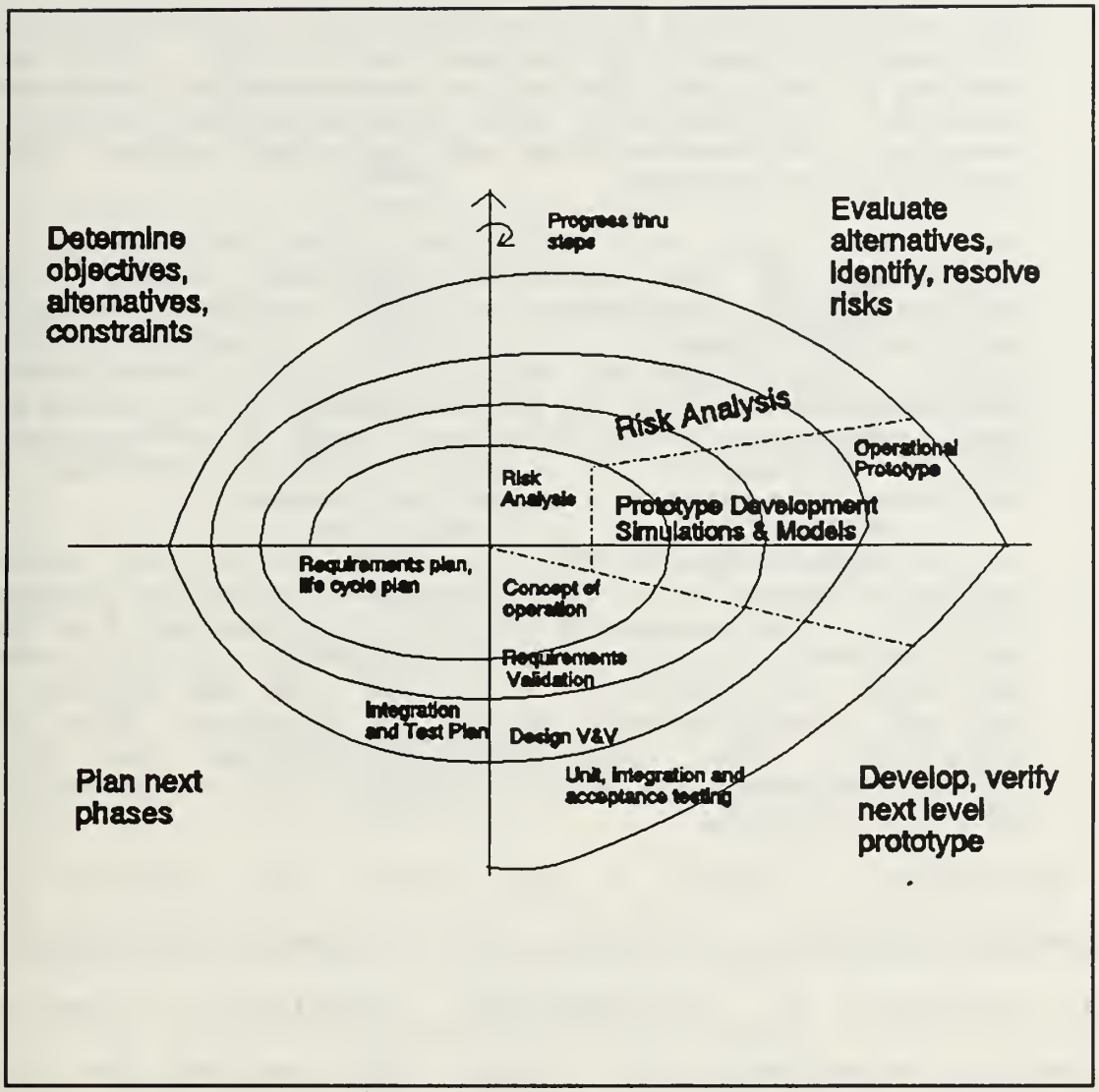


Figure 1 (Boehm, 1988)
Spiral software development model

Characteristics such as performance, functionality, and flexibility are defined in each cycle. Boehm [Ref. 4:p. 65] describes the next steps:

- The next step is to evaluate the alternatives relative to the objectives and constraints. Frequently, this process will identify areas of uncertainty that are significant sources of project risk. If so, the next

step should involve the formulation of a cost-effective strategy for resolving the sources of risks. This may involve prototyping, simulation, benchmarking, reference checking, administering user questionnaires, analytic modeling, or combinations of these and other risk-resolution techniques.

- Once the risks are evaluated, the next step is determined by the relative remaining risks. If performance or user-interface risks strongly dominate program development, or internal interface-control risks, the next step may be an evolutionary development one: a minimal effort to specify the overall nature of the product, a plan for the next level of prototyping, and the development of a more detailed prototype to continue to resolve the major risk issues.

- If this prototype is operationally useful and robust enough to serve as a low-risk base for further product evolution, the subsequent risk-driven steps would be the evolving series of evolutionary prototypes going toward the right in [Figure 6]. In this case the option of writing specifications would be addressed but not exercised. Thus, risk considerations can lead to a project implementing only a subset of all the potential steps of a model.

Noteworthy is that in each cycle the software is thoroughly analyzed and tested prior to further development. This technique of advancing more functionally complete prototypes supported by the spiral model reduces the risk associated with development of ill defined systems and promotes a product high in user confidence.

B. EXPERT SYSTEM LIFE CYCLE

The RAND Corporation [Ref. 5:p. 12] has proposed an expert system life cycle based upon the spiral model described earlier. Figure 2, adapted from Kameny et al, [Ref. 5:p. 12] describes a model of this life cycle.

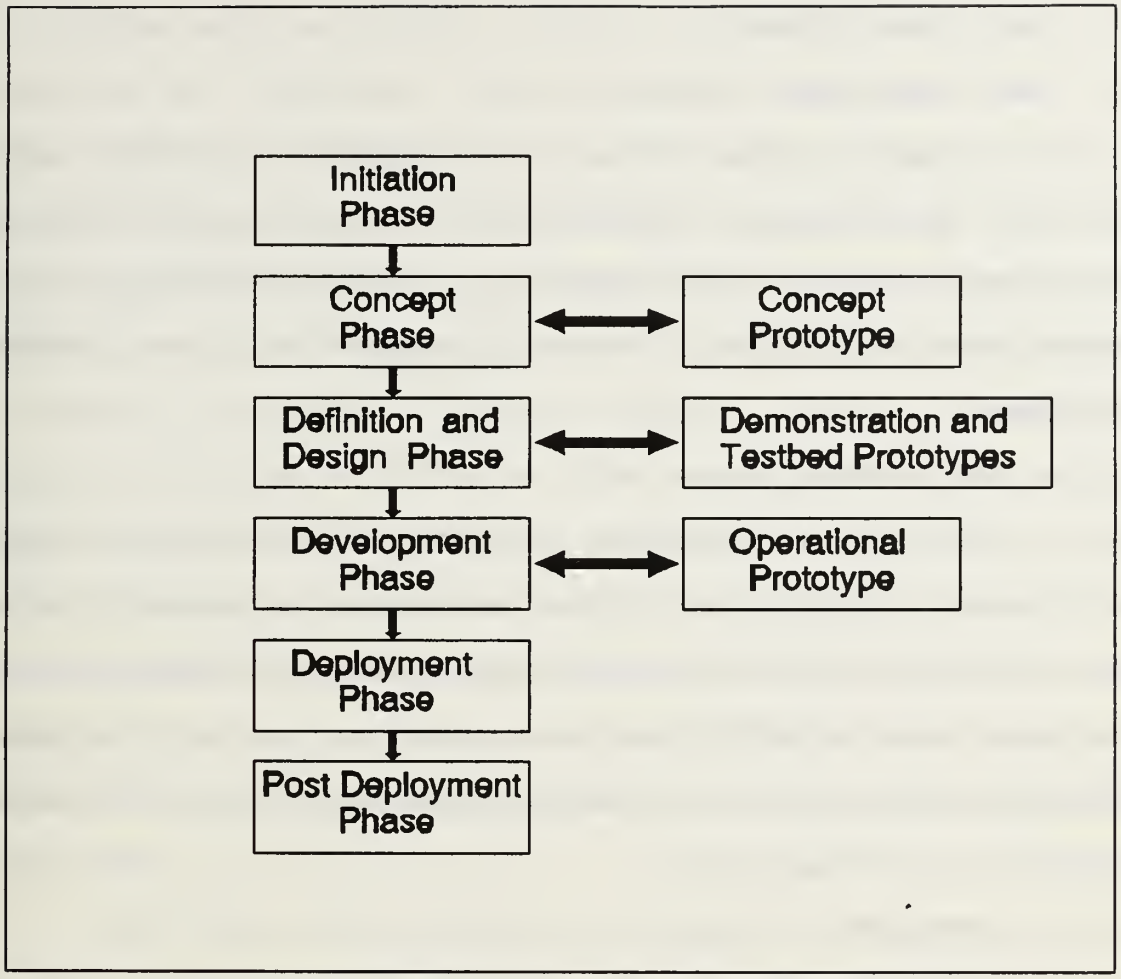


Figure 2
Expert system life cycle

This expert system development process is configured into the following phases:

- Initiation Phase
- Concept Phase (Concept Prototype)
- Definition/Design Phase (Demonstration Prototype and Testbed Prototype)
- Development Phase (Operational Prototype)

- Deployment Phase
- Post Deployment Phase

The expert system life cycle defined by this model involves four stages of prototyping: the concept (initial) prototype, demonstration prototype, testbed prototype, and operational prototype, to be discussed in the following sections.

1. Initiation Phase

This phase determines 'what is wanted' by both upper management and the user in the context of an expert system. The process begins with an initial screening to determine if the problem is suitable for development into an expert system. Once the initial screening has been completed, a more detailed analysis is accomplished.

2. Conception Phase

This phase decides how the problem is to be solved. A deeper understanding of the problem is learned through further research and development of a conceptual prototype, a first prototype in its roughest form. One of the most critical aspects of this phase is development of the initial test and evaluation plan, which will be used during the Definition/Design, Development, Deployment, and Post Deployment phases. The test and evaluation plan will need further refinement during the subsequent stages of the life cycle.

During initial development of the test and evaluation plan, test cases statistically representative of the problem

categories must be defined. These test cases must cover the spectrum of problems envisioned by the user, and be representative of all known categories in the expert's domain. Enough test cases must be available for testing from the initial prototype through acceptance testing by the user. The initial set of test cases developed must be extensive enough to cover the entire testing cycle. During subsequent prototype testing results are compared to those obtained after initial prototype testing, in an effort to determine what effect the new knowledge has within the expert system. This is not to preclude new tests from being added to the test case data bank as new knowledge is added. Rather, these new test cases will supplement those initial cases used throughout the life cycle.

The test and evaluation plan must also involve end-user participation at critical points throughout the process. These end-users must be identified early and be separate from those users working closely with the development team. The knowledge engineer needs to work closely with the end-users to develop evaluation techniques which determine how much benefit will be derived by the organization after implementation of the expert system. Acceptance testing must be defined in the initial test and evaluation report and used as a determinant as to whether further development in future stages should proceed (acceptable) or if refinement is required (not acceptable).

Timeliness measures, system correctness, and acceptable levels of degradation as the system grows in complexity, are tests for acceptance. Timeliness measures determine how long the user is required to input information in the system before a conclusion is reached. If the expert system requires more time to analyze the problem domain and search through the network than what the user is willing to allow, then the system will be doomed to failure. The user will determine it is easier to use conventional methods to reach a conclusion and not employ the expert system.

System correctness determines the level of acceptability from both the expert and user's viewpoint. Remembering that the expert is not always completely accurate, a determination must be made as to the level of accuracy of the expert system response. A certain level of accuracy must be acceptable to both the expert and the users.

Degradation measures how well the expert system responds after it has been running for a period of time and many user inputs have been introduced. Degradation may require a review of heuristics, redesign of the knowledge base, or a different selection of tools.

Testing and evaluation methodologies and results must be accurately recorded throughout the entire life cycle, then delivered to the maintenance organization during the post deployment phase. An accurate track history of system development, with test case results annotated in detail, will

greatly assist those responsible for maintaining the expert system.

In addition to preparing the test and evaluation plan, preliminary training is provided for the expert, selected end-users, and staff involved in development. Training should consist of, at a minimum, explaining the technology, providing on line experience with a fully functional system, and creation of a very small rule based system for demonstration purposes. This will encourage familiarity and reduce the perceived threat normally felt when acquiring new technology.

Once the initial prototype has been evaluated by the expert and determined to be satisfactory given the defined limits of acceptance, the project team is ready to move into the Definition/Design phase.

3. Definition/Design Phase

This phase consists of two prototyping efforts, the demonstration prototype and the testbed prototype. The demonstration prototype is the second of four prototypes developed during the expert system life cycle. Development of the prototype is accomplished with an expert system tool selected for its flexibility during design. The first step in the technical process of building the prototype is that of defining the technical plan. This effort includes describing the functionality of the prototype, which particular tests will apply from the test and evaluation plan, and obtaining commitments from the experts and selected end-users for

testing. Once testing begins, a predefined objective must be attained through each iteration. Required results, if not attained, force the knowledge engineer to reexamine the knowledge base, consult with the expert, and retest until satisfactory results are attained. The final testing of the prototype involves evaluation by a small group of users not directly involved in the prototyping effort.

Test cases should be evaluated by experts independent of the expert system development effort. Additional qualified experts should be made available to review testing results at the end of every iteration. The credibility of the experts used to evaluate test results adds to the credibility of the expert system.

The testbed prototype, the third of four prototypes, differs from the demonstration prototype in that the completed version will result in a stand alone system. Testbed implementation will actually use the expert system tool that will be used in the operational prototype phase. Application needs will be stressed rather than flexibility of the tool. Experience from building the earlier prototype is used. The purpose of the testbed prototype is to build upon the previous iterations and develop an expert system more meaningful to the user. Each iteration has a well defined objective. Testing of the prototype is similar to that of previous phases. The measure of the actual validity of the prototype, and the expert system itself, is in essence how well it captures the

knowledge of the expert and how well the system can explain its reasoning. There will be four sets of tests: [Ref. 5:p. 86]

1. Test cases from previous test and evaluation runs (including Demonstration Prototype test and evaluation runs) will be rerun to establish the fact that the new expert system supports the established test and evaluation baseline. It is possible that a major change will induce some test cases to fail (e.g., if a category has been removed). Any failures should be individually addressed by the knowledge engineer and expert.

2. New test cases will be run by the test and evaluation team (with the help of technical team members if necessary) and the responses recorded and analyzed with respect to the criteria. Special tests may need to be run to demonstrate the ability of the expert system to meet the performance requirements.

3. New test cases will be run by the expert and responses recorded and analyzed by the expert with respect to the criteria. If the rationale has not been provided, then the expert should judge how well the expert system handled the reasoning behind the response.

4. End-users will use the system to handle selected cases with technical staff help if necessary. Their performances will be monitored and reported by the test and evaluation staff. After using the system, the end-users will be interviewed about the problems they have and their suggestions for improvements. If possible, the end-user tests should be recorded using audiovisual equipment.

The test and evaluation sessions should be reported in a progress notebook. Updates for the test and evaluation plan need to be introduced. Updates include such things as new test cases or modifications of test cases already intro-duced with new values. Experiences from developing the testbed prototype may indicate a need for better criteria or for changes in the design or numbers of test cases. Sufficient test cases may have already been provided. However, unlike a

conventional system, where a correct answer exists for test cases, an expert system response may be considered adequate for each test case. The more varied the test cases are, the better the expert system can cover the entire problem domain, testing for resiliency at the edges. Test cases will cover those values frequently seen in the problem domain, those on the "boundary edge" which are infrequently used, and those values whose usage is never expected, in a determination of where the system will fail.

The final test and evaluation using the testbed prototype takes users unfamiliar with the project, gives them some end-user training, and haves them solve a set of problems using the expert system. Evaluation criteria must be reestablished beforehand. Results are diligently recorded, adding to the lessons learned in previous iterations and phases.

4. Development Phase (Operational Prototype)

The operational prototype, the last of the four prototypes developed, is a stand alone program using the actual expert system tools to be incorporated in the final system. The end product should meet all requirements expected of the fully developed expert system. Speed, robustness, clarity, the ability to explain its reasoning, and correctness are the performance requirements given detailed examination. A friendly user interface as determined by the user is

evaluated. Degradation must be examined after the system has been operating continuously for a long period of time.

Specific testing procedures are required of the operational prototype before a fully capable expert system can be deployed. Test cases run during testbed prototyping should be run again, with differences compared. Additionally, more complex cases than those used during earlier testing should be run. Correctness and timeliness of performance are evaluated.

Actual data from a current case in the real world is then tested on the operational prototype. Once again, correctness and timeliness of responses are evaluated by the reviewing experts. Recoverability is important. Failure can be introduced in many ways, and the system must be able to fully recover and function as before. Physical tests on the system include power outages and specific equipment failures. Errors introduced which the system needs to recognize and overcome include entering wrong data values, duplicating data, or intentionally leaving data out. Full testing and evaluation follows each malfunction, with results meticulously recorded by the knowledge engineer.

Testing for robustness involves leaving the prototype running for an extensive period of several days. Experts and end-users test the system exhaustively to determine if any degradation is existent and the extent of recovery. The purpose of this phase is to take the testbed prototype and incorporate it into an operational environment.

Once exhaustive testing of the operational prototype is complete, the expert system life cycle moves into the deployment phase.

5. Deployment Phase

During this phase the Operational Prototype developed in the previous phase is tested by the users at the user's sites. Training is conducted and the prototype is used under realistic conditions by all operators. This process of moving from the prototype version to the real life system may be a simple maneuver of changing a few parameters, to a more complex modification.

Deficiencies requiring extensive modifications discovered during operational prototype testing may cause the entire project to revert back to the testbed prototyping stage. Further development, testing and evaluation will be required before the deployment phase is reached again. The operational system, once completed beyond the prototype stage, is released to the user.

6. Post Deployment Phase

This phase takes place after the expert system has been turned over to the support organization during the Deployment phase. During it's first six months of operation, the expert system must be updated as necessary and observed for any degeneracy in results. A maintenance team will track the system, adding updates as required, and reporting to the expert system developers any problems encountered. Beyond the

six month post deployment phase, the expert system is normally turned over to the maintenance team for any upgrades required in the future.

C. SUMMARY

The spiral model, as developed by Boehm, has proven useful in expert system development. Its design is based on the incremental development method typical of knowledge based systems.

The expert system development process consists of several phases: the initiation phase, concept phase, definition/design phase, development phase, deployment phase, and the post deployment phase.

Chapter III examines the validation, verification, and testing procedures used for development of expert system testing plan.

III. VALIDATION, VERIFICATION AND TESTING OF EXPERT SYSTEMS

Validation, verification, and testing (VV&T) are formal methods used to determine both the correctness of a program and whether the program will satisfy user requirements. A shortfall of VV&T requirements reduces the likelihood of acceptance by users. Several questions may be raised. How is the user to know when the expert system is correct? What are the experts acceptability standards for the expert system? Is the proper advice offered?

Testing and validation methods for conventional software are time consuming, difficult, and not always correct. Correctness in this case refers to desired software specifications being included, as opposed to correctness with no chance of error. Traditional software validation is incorporated into usage of the waterfall model, until recently the most prevalent model of software development. It involves the entire software life cycle, and is not oriented towards the incremental prototype stages of knowledge based systems development. Conventional software projects have precise, rigid requirements for development, whereas knowledge based systems, by their nature, often have less precise design specifications, especially during early development. This can be overcome by incorporation of the Boehm spiral model of rapid prototyping as described in Chapter II. Newer

specifications are added as the prototype evolves from simplistic to more complex.

Knowledge based systems are non-procedural, not containing modules analogous to those in conventional software. Inputs are not necessarily mapped to specific outputs. Combined with less than precise design requirements, testing of a knowledge based system becomes inherently more difficult.

To date, research on testing methodologies for knowledge based systems has not approached that of conventional software systems.

This chapter will discuss validation and verification (V&V) techniques for expert system development. V&V in the first generation rule set will be examined, followed by further V&V procedures as incremental prototypes are developed. Finally, testing of the entire expert system prior to final delivery is discussed. Quantitative and qualitative measures are introduced, with the final sections of the chapter discussing development of a successful test plan for expert systems.

A. PURPOSE OF VALIDATION AND VERIFICATION

Validation refers to the process of determining whether the expert system is 'correct'; that is, whether it meets the level of accuracy as required by an acceptable set of standards. Validation substantiates whether the right system has been built [Ref. 6:p. 29]. It may be considered a "live

activity" in which the software is tested under both laboratory and operational conditions.

The verification process determines whether the completed system has been correctly implemented according to predeveloped specifications. Often, verification is nothing more than a paper drill, where specifications are read, compared, and cross referenced. This research has found a paucity of literature and an indication of limited use of V&V in expert systems development.

Green and Keyes [Ref. 7:p. 39] list the following benefits if V & V were applied to expert system development:

- Expert systems would be fielded with less risk of software failure. This would promote the use of expert systems technology in mission critical systems.
- Organizations wary of expert systems because of the lack of V & V would be more inclined to employ this new and desirable technology.
- Experimental application of V & V to expert systems would permit the development of effective V & V methodologies for expert systems.

What is needed is a way to make progress towards developing an effective V & V methodology. First and foremost, a specifications document must be written and maintained throughout the development of the expert system.

The expert system developer must have a clear idea of what the expert system is expected to accomplish. Once these expectations are defined in the specifications document, design, testing, and V&V for the expert system commence.

B. VALIDATION OF EXPERT SYSTEMS

To a large extent, the quality of the expert system is determined by validation of the software. Quality is predefined in the specifications document. Accuracy of the final recommendation, time required to reach this recommendation, and possible system degradation are all measures of quality. The expert system must be tested exhaustively prior to being put into operation by the end user. Validation is not to be considered a once and for all check of the system to see if it meets a given specification. Specifications may change during the evolutionary cycle of the expert system development. Validation must follow each of the specification changes.

Defining the quality of an expert system, and thereby the criteria by which it is to be judged, is taken into account when drawing up system specifications. For instance, in the development of ESAAMS, how is the new knowledge to be added and the old knowledge maintained? Can the new knowledge be linked to the old knowledge in a consistent fashion recognizable by the expert system? During various stages of the development cycle, the following factors should be considered [Ref. 8:pp. 174-175].

1. The degree of correctness of the system's conclusion. Was its advice good or not? Acceptable is a reasonable response, one that may not be entirely correct, but is certainly not wrong.

2. Sensitivity. To what extent is the correctness and/or precision of the outputs affected by the precision of the input information?

3. The precision and correctness of any intermediate conclusions. Intermediate conclusions are necessary for tracing the logic used within the network. Accuracy of these intermediate conclusions determine how precise the final conclusion will be.

4. The precision and economy with which the reasoning is carried out. This includes the number of steps to reach the conclusion and the amount of data required as opposed to that required by the human expert, and other factors that bear upon acceptability of the system.

5. Response time. This is the total time taken by the system in giving its conclusion.

6. Robustness. Resilience under variations in such factors as the environment and the quality of the users.

Once these factors are incorporated into the specifications document, verification of the expert system is straightforward. Correctness of intermediate and final conclusions, sensitivity, response time, and robustness are measured against the predetermined measurement. When specification requirements are not achieved, further incremental development and testing is necessary for deliverance of an acceptable product.

C. VALIDATION AND VERIFICATION DURING SYSTEM DEVELOPMENT

Prerau [Ref. 9:p. 312] describes the knowledge acquisition cycles as Elicit-Document-Implement-Test. Repeated validation testings occur in the life cycle of the evolving expert system. During the testing stage, deficiencies in the expert systems knowledge base are detected by comparing results of a test case against either documented knowledge or the human expert. Each time the test case is run, the new incorporated knowledge is tested with other previously tested

knowledge. Should the program abort, a major error is suspect. When the program runs to completion, but results disagree with those of the expert, the new knowledge, as defined by rules, is suspect.

D. FIRST GENERATION RULE SET

The first generation rule set will typically consist of a few dozen rules. Main objectives of this early prototype are to "reexamine the original objectives and more precisely determine the problem domain, and establish the degree of detail desired in the system." [Ref. 10:p. 44]

Accuracy of the system is quantitatively measured by comparing the number of correct predictions with known data. Statistical inferences may then be drawn using accumulated data. Percentages of correct answers should approach those of the human expert. A simple percentage of right/total may be sufficient. An alternative method would be to weight certain conditions which are more important to the expert system developer. For example, certain test cases more representative of the expert system environment are given layer relative weights than others. Statistical inferences measuring all system responses favor the test cases using the largest relative weights.

The degree of precision required is another issue during the initial testing stage. "Precision may be measured as the capacity of the knowledge base to predict, diagnose, classify,

or monitor within a specified statistical confidence internal." [Ref. 10:p. 44]

Once the prototype has been tested, the resultant rule set must meet certain specification criteria predetermined in the test plan. As rules are expanded or modified within the knowledge base, performance standards are brought in focus and described, in preparation for development of future prototypes.

E. EXPANDING THE KNOWLEDGE BASE AND TESTING

Once a prototype has been deemed successful by meeting preestablished criteria, the knowledge engineer extends the knowledge base creating yet another prototype. The knowledge engineer selects some tests cases with a predetermined solution as determined by the expert, and checks for further consistency in the knowledge base.

Old test cases used during initial prototype development plus additional test cases tailored to the new prototype are introduced into the testing cycle. This process of saving old test cases and introducing new test cases continues throughout each prototype iteration. If the test cases give rise to problems, the knowledge engineer analyzes the new rule set with the expert in order to locate the difficulty. Modifications are made, and the knowledge engineer continues to test the system until the expert is confident the predetermined accuracy level has been reached. If no more problems are

located from the set of trial runs, then the current edit or extension of the knowledge base is deemed successful.

This process of rapid prototyping, testing and editing continues until the full expert system has been developed. Once complete, the expert system is delivered to the user, and operational testing begins.

After development is complete, the expert system must be validated against the outside world. In the case of absolute validation, correctness is easy to measure. A variety of actual test cases are run, then measured against the known response.

When absolute certainty to an answer is not known, an expert or team of experts must generally agree on the correctness or optimality of a result. When no clear notion of a perfectly correct solution exists (such as prioritizing an aircraft maintenance schedule), then the experts must generally agree on whether a decision is optimum, reasonable, or at least acceptable. When experts strongly disagree as to what the results should be, then the expertise of a single expert or small group of experts are used as a baseline to measure results against.

Validating the expert system using an expert or group of experts not used during the developmental stages has the benefit of removing possible biases. Subtle variances in reasoning may also be detected. Once results are accepted by

the outside group of experts, a great deal of credibility is added to the expert system.

Two testing methodologies, qualitative (Turing test) and quantitative (paired t-tests) are available to assist the knowledge engineer in determining the accuracy of the expert system.

1. Turing Test

An effective testing methodology used to assess the validity of correctness is to use the Turing test. Turing tests validate expert systems by evaluating human expert performance and system performance without knowing the subject performers identity. [Ref. 11:p. 86] A user, or another expert, is shown results from questions posed to either an expert or an expert system, without revealing the identity of the answering mechanism. When the expert or user assessing the solution cannot distinguish between the expert and the machine, then the validity of the program is deemed acceptable as to how an expert would answer the question.

2. Quantitative Validation

Quantitative validation employs statistical techniques to compare expert system performance against either test cases or human performance [Ref. 11:p. 87]. A quantitative method applied to measure the consistency of responses between the machine and the expert is the paired t-test.

A confidence interval for one or more measures is established, where results are compared against an acceptable

performance range, or a formal hypothesis test is used. The hypothesis test criteria are: [Ref. 11:p. 87]

H_o: The expert system is <u>valid</u> for the acceptable performance range under the prescribed input domain.

H_1: The expert system is <u>invalid</u> for the acceptable performance range under the prescribed input domain.

O'Keefe [Ref. 11:p. 87] proposes using paired t-tests to compare the difference between observed test results. Differences (D_i) are measured between results gathered from the expert system's performance against those of the human expert or known results. Once rated, a difference between the two is recorded where $D_i = X_i - Y_i$.

X_i are the expert system's results, and Y_i are either known results or results from human expert performance. For n test cases, there will be n observed differences, D_i to D_n. For these differences, the following confidence interval is produced:

$$\bar{d} \pm t_{n-1,\, \alpha/2}\, S_d / \sqrt{n}$$

where $\bar{d}$ is the mean difference, S_d the standard deviation, and $t_{n-1,\, \alpha/2}$ the value from the t distribution with n degrees of freedom. When zero lies within the confidence interval, the system's performance, H_o, is deemed acceptable.

F. OPERATIONAL TESTING

The final part of validating the system consists of operational testing. Operational tests are an important part of the total evaluation process. Successful results will

almost certainly convince potential users of the merit of the system, providing sound reasoning for continued development and successful deployment.

Users should be properly trained in operating the expert system, understand its functionality, and be able to use its results correctly with a minimum of effort. One of the most crucial aspects of development of the expert system is providing a user friendly interface. Only through thorough operational testing, observation, and careful recording of data is this certified.

The remaining sections of this chapter discuss what is to be included in a test plan, followed by a checklist of recommended procedures for testing an expert system.

G. TESTING AND TEST PLANS

The purpose of testing is to ensure that the expert system can accurately solve the particular problem or class of problems for which it was designed. Crucial to acceptance by the user community is for the system to "prove itself," i.e., to be beneficial for the purposes intended. Experts are chosen to solve particular problems because they have a track record of known successes. They have proven that they can be trusted with the decision making requirements. Conversely, an expert system also needs to prove that it can be trusted. A comprehensive and multifaceted test plan with a rigorous set of specifications ensures the confidence level required by the end-user. It is impossible to test every possible case.

There are 2^n power different paths a system may take when exploring all possibilities in a network. This number grows exponentially larger as more knowledge is added to the system or the number of rules increases.

Values within, at the boundary edge, and outside the set of constraints and capabilities built into the system need to be checked. Valid input should lead to an expected conclusion acceptable to the expert. Values "on the edge" are checked to determine if the expert system will fail, producing an unreasonable conclusion, or if the expert system will request more information from the user. One thing the expert system must do is immediately reject invalid inputs. Errors spotted must result in notification of the user.

Knowing what the input is and what the expected outputs should be is a fundamental concept in software testing. Expert systems differ from conventional systems in that mistakes are made by the expert system as well as the expert. Test cases are selected based upon known output values. Successful testing is accomplished when the expert agrees with the solutions reached by the expert system.

A difficult question often posed during expert system testing is, "How much testing is enough?" Unfortunately, no simple answer exists. The test and evaluation plan developed during the earliest development phases must address this topic.

For example, a determination as to how many test cases must be developed, and how many variations of each test case are to be used should be provided. A recommendation for development of ESAAMS is that test cases be derived from actual maintenance forms processed over a one week period. Once the maintenance actions have been identified, variations of the test cases are identified and tested. Further discussion is included in Chapter V. It is envisioned that a relatively small number of test cases will be executed in proportion to the number of variations possible. Having a sound testing plan in place before actual testing begins is a key to successfully developing an expert system.

Different aspects of the system as discussed in the following sections, must be tested and weighed accordingly. The evaluation criteria must be measurable. It may be either objective or subjective.

Objective criteria are easily measurable. A known answer exists and results from test cases are identified as either right or wrong. Subjective criteria measurements are more difficult to judge. The experts or users are asked to decide the quality of a facet of the expert system based on their opinion. Opinions may vary widely on how correct a conclusion may be, or how friendly an interface is. The following are a list of recommended tests [Ref. 12:p. 6].

1. Structural Tests of the Knowledge Base

These tests are concerned with the underlying structure of the knowledge base. Structure deals with how rules and facts are assimilated within the knowledge base. The logical consistency and functional completeness of the rule base will be checked. Tests for logical consistency are aimed at finding and correcting redundant rules, subsumed rules, conflicting rules, and unnecessary if conditions. Tests for logical completeness are used to find unreferenced attribute values, illegal attribute values, unreachable conclusions, and deadends in the knowledge base. Explanations and examples of these terms are discussed in Chapter IV.

2. Content Specific Tests for the Knowledge Base

In this second type of testing, the expert or team of experts evaluate the accuracy of the embedded knowledge within the expert system. The domain expert assesses the adequacy of the facts in the knowledge base, the adequacy of the actual rules, accuracy of the knowledge representation, and assessments of modifications made to the knowledge base.

An important criterion for judging the expert system is its accuracy. When expert systems are to be used as experts, or expert advisors, then they should be expected to perform at the same level or greater than the expert. Just as the expert may be infallible with regards to accuracy, so may be the expert system.

To overcome this shortfall, when testing the expert system for completeness of results, the system developer, with the help of the expert, should first define the level of desired performance criteria. Enough sample cases must be run to test whether the system meets the minimal performance criteria.

Associated with the level of accuracy required is the issue of completeness (i.e., containing the same level of knowledge as the expert). As the expert system grows in size, the more complex it will become, with the chance for errors growing proportionally. Since the expert system may never be complete, a better strategy is to define completeness required at the onset of the developmental process acceptable to the user. The developer sets a carefully predetermined figure for what is expected of the system, then measures the developed system against this figure.

Completeness is a difficult topic to address, as the expert system may never be totally complete. There will always be room for more knowledge, more facts and heuristics from the expert. Using the predetermined figure of what is expected of the system, once a specific knowledge level as determined by the specifications document is reached within the expert system, the system is considered complete.

3. Performance Tests

Performance tests, the third set of tests, determine how well the system carries out its designated function.

Tests may be divided between those where a known answer exists, or those where judgment of the experts is the best indicator of the most correct solution. Quantitative analysis, as described earlier, is the best measurement of performance. Run time efficiency is assessed after other critical elements, such as correctness of advice given, have been tested and validated. The user may elect not to use the system because it is either too slow to be effective, asks too many repetitive questions of the user and thereby delays the response time for the advice, or the system cannot interact effectively when accessing an external database.

Fortunately, solutions do exist to remedy these problems. Hardware configurations, such as adding additional RAM or increasing CPU speed, is possible to increase response time.

Large expert systems such as Neuron Data's Nexpert Object operate most efficiently on a 386 based machine with 4 megs of RAM. Although smaller machines operating with a 286 microchip can handle some expert systems, delays while the expert system searches the network attempting to reach an intermediate or final conclusion may seem excessive to the user. Design specifications, whereby the most frequently accessed information is stored in the expert system itself, will increase the speed of the system.

4. Usability Tests

User friendliness is often a key factor in the success of an expert system. No matter how accurate performance measures are, how consistent or complete the system may be, or how reliable the expert system is, all development may be in vain if the system cannot be easily accessed or convey its knowledge or expertise effectively. Information must be clear, understandable, and easy to follow, with user input and output adapted to the skill level of the user. There are two types of user friendliness testing: subjective and objective. Subjective tests involve direct query of the user, normally by questionnaire. Users are asked such questions as the following: [Ref. 13:p. 227]

1. Was the recommendation correct?
2. Was the response format acceptable and efficient?
3. Were the system recommendations clear and useful?
4. Was the reasoning explained at a level that you could easily understand?

Objective usability testing does not query the user directly, but rather involves close observation of the participant as he interacts with the expert system. Certain measurable factors may be obtained, such as:

- response time to answer a question;
- number of keystrokes used to enter a response;
- number of times explanation of reasoning was used or a help facility invoked;
- degree of immediate productivity perceived on the part of the user after a consultation.

The above tests, structural tests of the knowledge base, content specific tests for the knowledge base, performance tests, and usability tests, cover the complete range of testing an expert system.

Independent testers both within and outside of the organization play a key role in validating some expert systems.

H. INDEPENDENT VALIDATION AND VERIFICATION (IV&V)

Most expert systems are rather small in nature (< 500 rules), with V&V accomplished by the knowledge engineer and expert. Some large organizations may have separate IV&V teams available for the testing of expert systems.

Many larger expert systems subcontracted out by the Department of Defense (DOD) may require the services of an IV&V team. Some of these expert systems offer advice for weapons systems. Incorrect advice offered during a crisis situation may prove catastrophic. The level of IV&V conducted should be consistent with the amount of risk taken in using the expert system. Normally independent IV&V should not duplicate original V&V.

The United States Army has an IV&V team for testing expert systems at the Electronic Proving Ground, Fort Huachuca, AZ 85613-7110 (Attn: STEEP-ET-S). Their IV&V team consists of four members responsible for testing Artificial Intelligence and knowledge based systems throughout the Army. Primary testing is accomplished on site. Testing the resiliency of

the software, accuracy of advice offered, user-friendliness, and the user interface are the major areas examined. As expert systems become more prominent throughout DOD, it is expected more IV&V teams will be necessary.

The next section presents a checklist of key points for validating and verifying expert systems.

I. VALIDATION & VERIFICATION CHECKLIST

Using a checklist for evaluating an expert system ensures that all key points are covered. One of the best examples of a checklist for testing and evaluating an expert system is found in Prerau [Ref. 9:pp. 312-313] and is provided below:

- Use the cycle of Elicit-Document-Implement-Test not only for knowledge acquisition but also as a way to test the evolving expert system program continually. .
- In a domain where the correctness of an expert system's results can be determined absolutely, measure the competence of the system by the degree of its agreement with the known correct results. To determine the overall worth of the system, this measured competence should be compared, in most cases, not against a standard of perfection but against the proficiency of typical domain practitioners.
- In a domain where experts usually agree, evaluate the system by comparison against human experts.
- In a domain where experts often disagree strongly and irreconcilably, compare the expert system's results against the results of the project's expert(s) and be happy with a system that has expertise close to that of the project expert(s).
- When the domain allows, utilize for system evaluation an expert or experts not associated with the project, as long as the gains in impartiality of evaluation and credibility of result outweigh any difficulties and costs.

- If the domain allows, use multiple experts for system evaluation when the gains in credibility of result outweigh the problems that occur when the evaluating experts disagree.

- Use meetings with consulting experts to evaluate system results and also to evaluate the detailed reasoning and internal processes of the system.

- When domain and organizational conditions permit, test the expert system in the field. Set up a field trial to evaluate the expert system's performance under actual operational conditions, or test the system during its initial routine production use.

- If live field testing cannot be performed, consider running a parallel field trial, where the system is run on real data but in a nonoperational setting that parallels actual operation.

- Control field testing carefully to ensure that procedures are followed correctly and that field personnel understand the expert system and know how to use it. Try to ensure that no factors unrelated to the expert system's competence can be the cause of poor results.

- Give careful consideration to the types of data that will be collected during a field test. They should accurately reflect the performance and other important factors related to the expert system and should be convincing to others.

- Set up mechanisms that allow the gathering of the accurate field test data while imposing as little burden as possible on users.

- Verify that the program accurately implements the acquired expert knowledge. The knowledge acquisition process by its nature will likely result in a final expert system program that agrees very well with the knowledge documentation.

- Verify that the implemented expert knowledge contains no internal errors (independent of the completeness or correctness of the knowledge itself), such as redundant rules, sets of circular rules, and illegal slot values. Utilize automated checking systems if available.

- Verify that the base program that implements the knowledge engineering paradigms operates correctly. This aspect of verification may be minimized if the

project is utilizing a standard, commercial software tool in wide use.

- Put the amount of effort into system evaluation that the particular system warrants. Invest substantial effort in a large-scale evaluation if errors by the expert system would be disastrous or if knowledge of the exact competence level of the system is critical. Invest less evaluation effort if system errors are not crucial and any performance close to an expert's is valuable.

- Set standards of evaluation for the expert system based on domain requirements. In critical applications, standards should be very high. When errors are not costly, consider using lower standards to gauge success.

J. SUMMARY

This chapter has discussed the V&V of expert systems in detail. V&V of an expert system is difficult. Certain steps must be taken to ensure accurate system development. First, a specifications document describing exactly what the expert system is expected to accomplish must be completed. Then a testing and evaluation plan must be conceived. Results from testing the expert system must be meticulously recorded in the progress notebook.

A testing plan must be designed early in the expert system life cycle. Enough test cases must be available for testing each prototype iteration. As system development progresses, test cases used in previous prototype testing will be rerun to determine what effect changes to the knowledge base have caused.

A checklist for testing and evaluating expert systems summarizes key points listed in the chapter. The following chapter will discuss knowledge base debugging methodologies.

IV. KNOWLEDGE BASE DEBUGGING

This chapter describes procedures for debugging the knowledge base of the expert system. Neuron Data's Nexpert Object is examined, followed by a description of two external programs useful as debugging tools for expert systems.

Knowledge base debugging tests the rules in the knowledge base for consistency and completeness. Consistency checks determine whether there are any redundant rules, conflicting rules, subsumed rules, unnecessary IF conditions, or if any circular rule chains are present. Completeness checks look for unreferenced or illegal attribute values, dead end goals, unreachable conclusions and dead end goals.

Some expert system shells only have syntax checkers built in, while more complex shells like Neuron Data's Nexpert Object offer a wider range of debugging tools. Two external programs, VALIDATOR and CHECK, have been developed as debugging tools to assist in checking expert system shells for knowledge base errors.

A. CONSISTENCY CHECKS

The following are several types of inconsistencies which may be found in a knowledge based system.

1. Redundant Rules

This situation is encountered when two rules succeed in the same situation and give the same results. It happens

when the IF parts of two rules are equivalent, and one or more conclusions are also equivalent. The IF parts of both rules must have the same number of conditions, and the condition in one rule is equivalent to the condition in the other rule. For example [Ref. 14:p. 71]

a. IF X has a hoarse cough, AND
   X has difficulty breathing
   THEN type-of-disease is CROUP.

b. IF Y has difficulty breathing AND
   Y has a hoarse cough
   THEN type-of-disease of Y is CROUP.

X and Y represent variables that will be attached to a person in the database. The rules would be redundant no matter what the order of the IF conditions. Differing variables make no difference.

Logic may not be affected in the knowledge base, but redundant rules may hinder the efficiency of the system, slowing it down as it searches for the optimal solution.

2. Conflicting Rules

Two rules are conflicting when they succeed under the same set of circumstances yet reach different results. The IF part of the two rules must be equivalent, yet results are contradictory.

For example: [Ref. 14:p. 71]

a. IF X has a hoarse cough, AND
   X has difficulty breathing

THEN type-of-disease of X is CROUP.

b. IF X has a hoarse cough, AND

X has difficulty breathing

THEN type-of-disease of X is BRONCHITIS.

Results may be disastrous. Given a similar set of circumstances, the expert system may give conflicting advice.

3. Subsumed Rules

One rule is subsumed by another if the two rules have the same conclusion, but one contains additional constraints for the situations in which it will succeed. For equivalent conclusions, one IF statement must contain more conditions to succeed than the other.

For example [Ref. 14:p. 71]

a. IF X has flat pink spots on his skin AND

X has a fever

THEN type-of-disease of X is MEASLES.

b. IF X has flat pink spots on his skin

THEN type-of-disease of X is MEASLES.

Whenever the more extensive IF statement in one rule succeeds, the other rule will automatically succeed. This produces a redundancy in the system.

4. Unnecessary IF Conditions

Two rules contain unnecessary IF conditions if the rules have the same conclusions, an IF condition in one rule is in conflict with an IF condition in the other rule, and all other IF conditions in the two rules are equivalent.

For example: [Ref. 14:p. 72]

a. IF X has flat pink spots on his skin, AND
   X has a fever
   THEN type-of-disease of X is MEASLES.

b. IF X has flat pink spots on his skin, AND
   X does not have a fever
   THEN type-of-disease of X is MEASLES.

Since both IF statement constraints conflict, yet reach the same conclusion, the unnecessary portion of both IF statements must be removed.

5. Circular Rules

Circular rules exist when a cyclical pattern forms when certain rules are chained together.

For example: [Ref. 14:p. 72]

a. IF temperature of X > 100 (in Fahrenheit)
   THEN X has a fever.

b. IF X has a fever, AND
   X has flat pink spots on his skin
   THEN type-of-disease of X is MEASLES.

c. IF type-of-disease of X is MEASLES
   THEN temperature of X > 100 (in Fahrenheit) given a goal of:
   type-of-disease of patient is measles.

This set of rules sets up an infinite loop. System efficiency is degraded, and dependent on the type of circular loop imposed, may eventually lock up. This type of logical

error is most difficult to detect and remove, and requires careful analysis of the knowledge base by the designer and expert.

B. COMPLETENESS CHECKS

The second test of rules in a knowledge base deal with completeness. A theoretically complete system captures all of the experts knowledge within a narrow domain. Limitations of the experts time, size of the rule base, and cost considerations preclude obtaining this knowledge. Given predetermined specifications describing the knowledge to be captured, completeness is still difficult to achieve. This happens because the knowledge acquisition process often leaves gaps in the knowledge base between the expert and the knowledge engineer attempting to capture his expertise. Additionally, as the knowledge base grows larger, it becomes impossible to check every possible combination of rules to ensure completeness. Below are four cases of attribute values which must be removed to reduce complexity within the knowledge base.

1. Unreferenced Attribute Values

This condition occurs when attribute values in the knowledge base are not covered by any IF conditions in the set of all possible rules. For example [Ref. 14:p. 72], suppose the attribute TEMPERATURE has the following range of values {high, normal, low}. If "high" and "normal" are specified by

IF conditions in the rule base, but no condition for "low" is made, then that value is considered unreferenced.

The knowledge engineer must then determine if a rule is missing which would include the "low" value, or if the "low" value should be removed from the set of values in the knowledge base.

2. Illegal Attribute Values

An illegal attribute value is similar to an unreferenced attribute value. It refers to an attribute value that is not in the set of legal values in the knowledge base. Quite often, a spelling error is the culprit. However, attribute values may be named differently, causing the problem.

For example [Ref. 14:p. 72]:

Suppose as before, TEMPERATURE has the values {high, normal, low}. If the rule is written as:

IF temperature of X is very high . . . or . . .

THEN temperatures of X is medium.

Both "very high" and "medium" are illegal attribute values, unrecognizable by the system.

3. Unreachable Conclusions

In a forward or backward chaining expert system, the conclusion of one rule should either attain a goal or lead directly to an IF statement in another rule.

For example [Ref. 14:p. 72]:

IF temperature of X > 100 (in Fahrenheit)

THEN X has a fever.

Should the condition of X not reach a conclusion or lead directly into another rule, then the conclusion of the rule is unreachable.

4. Dead End Goals

To achieve a goal (or subgoal) in a goal driven system, either the attributes of the goal must be askable (user provides needed information), or the goal must be matched by a conclusion of one of the rules in the rule sets applying to the goal [Ref. 14:p. 73]. Should neither requirement be met, a dead end goal has been reached.

C. NEXPERT OBJECT

The ESAAMS project initial prototype is using Nexpert Object. Nexpert Object has several facilities for detecting errors in the knowledge base. A rule network navigates the knowledge base, assisting the system designer in locating inconsistencies. A strong syntax checker has an automatic format checking mechanism which prevents the developer from saving (compiling) a rule with a wrong format in the system (missing arguments, wrong type of arguments, etc.). Navigation breakpoints allow the system to stop at each defined point and evaluate the reasoning process used. A short discussion follows of its VV&T attributes.

1. Rule Network

Nexpert Object's rule format is very powerful. Rules are not defined as "forward" or "backward", but depending on

the application when the rule base is consulted, the given rule may be processed (fired) from either direction. Thus, during a given session, a rule may be fired either forwards or backwards.

Links between the rules describe the structural relationships of the knowledge. With multiple rules creating a very complex knowledge base, NEXPERT OBJECT's NETWORKS gives the designer visual access to the interrelationships among the rules and objects. When the designer wishes to inspect a specific rule, object, or relationship, he need only "click" on the desired item in the network using the mouse pointer, then begin navigating through the different levels. Examining items on the microscopic level allows the designer to concentrate on a given topic of interest without losing the global, overall picture.

Networks are the fundamental tools for checking and discovering inference and inheritance inconsistencies by means of visual representation.

2. Syntax Checker

Nexpert Object's syntax checker determines if static errors are present. A rule editor screen provides a visual environment for the knowledge engineer to input domain information. A format checking mechanism prevents the developer from saving a rule with a wrong format (i.e., missing arguments, wrong type of arguments, etc.). The system will not allow compilation of an incorrectly formatted rule.

When the incorrect or incomplete rule is entered into the system, the incorrect box is identified for the developer to take corrective action.

3. Breakpoints

Visual breakpoints allow the inference engine to stop at certain points of evaluation (after firing a rule, after evaluation of a condition), allowing the developer to study the particular state within a reasoning process. Breakpoints are used to evaluate rules, hypotheses, conditions, actions, objects as they change, classes, slot values, properties, and methods.

Breakpoints, when combined with the network feature, assist the developer in locating redundant rules. When searching for conflicting rules and unnecessary IF conditions, all rules will fire until the stated constraints are found. The system will stop until these conditions are removed.

4. Dead End Goals

Dead end goals are virtually impossible not to notice. Every left hand side (LHS) condition is executed. When a necessary value is not found, the system attempts to inherit it from the parent. The next step is to backward chain, then reattempt to inherit a value from the parent. If this fails, the system will ask for a value from the child. Finally, the system will simply request a value for the goal from the user. Obviously, this situation creates excessive solution processing and must be avoided.

5. Summary

NEXPERT OBJECT, using the NETWORKS and BREAKPOINTS features, assists the knowledge engineer in locating redundant and subsumed rules. However, this is a trial and error methodology which may not locate all of the above mentioned defects in the network. Conflicting rules and unnecessary IF conditions will cause a system stoppage when invoked. Circular rules will cause the system to enter an infinite loop. The knowledge engineer then begins a search of the rulebase in an effort to discover the defective rule(s).

NEXPERT OBJECT's syntax checker does not allow for static error inputs. Erroneous inputs are flagged during rule insertion, allowing the knowledge engineer a chance to reinsert them correctly. Once the syntax checker allows a value to be input, NEXPERT OBJECT has no capabilities for conducting completeness checks.

A formal methodology is required to ensure consistency and completeness within the knowledge base.

D. EXTERNAL PROGRAMS

Two knowledge base programs for assisting the developer have recently entered the marketplace. These programs, CHECK and VALIDATOR, will briefly be discussed below.

1. CHECK

CHECK is a program designed to check a knowledge base for consistency and completeness. It was built for the Lockheed Expert System shell (LES), (Lockheed Research and

Development, Palo Alto, CA) but may be applied to many rule based systems [Ref. 14:p. 69].

CHECK identifies inconsistencies in the knowledge base by searching for redundant rules, conflicting rules, subsumed rules, unnecessary IF conditions, and circular rule chains. Checking for completeness is done by looking for unreferenced attribute values, illegal attribute values, dead end IF conditions, dead end goals, and unreachable conclusions. Gaps in the knowledge base which may have been overlooked by the expert and knowledge engineer may also be identified. CHECK also generates a dependency chart which shows the dependencies between rules and goals.

2. VALIDATOR

VALIDATOR (BICS, 1622 W. Monten Egro, Tucson, AZ 85704) checks for syntax and semantics errors, alerting the knowledge engineer. It consists of six modules; a pre-processor, syntax analyzer, syntactic error checker, debugger, chaining thread tracer, and knowledge base completeness module.

Tested on 67 various expert systems, VALIDATOR flagged many inconsistencies. Potential mistakes flagged by VALIDATOR fell into nine categories: illegal use of reserved words; rules that could never fire (both backward and forward rules); unused facts; unused questions; unused legal values; repeated questions; multiple methods (including expressions that appear in questions and facts, questions and conclusions, and facts

and conclusions); rules using illegal values; and incorrect instantiations. [Ref. 15:p. 48]

VALIDATOR was designed to make the task of testing expert systems easier while building confidence in the expert system design. Both have been accomplished.

E. SUMMARY

We have looked at several methodologies and tools for debugging a knowledge base in this chapter. Knowledge base debugging emphasizes checks for completeness and consistency. Consistency checks search for redundant rules, conflicting rules, subsumed rules, unnecessary IF conditions and circular rules. Completeness checks search for unreferenced attribute values, illegal attribute values, unreachable conclusions, and dead end goals.

Neuron Data's Nexpert Object has several key features for debugging its knowledge base: a rule network, syntax checker, and breakpoints.

Two external programs for debugging knowledge bases are available: CHECK and VALIDATOR. Each program described searches for inconsistencies in the knowledge base and flags them for corrective action by the knowledge engineer.

Techniques for debugging knowledge bases discussed within this chapter are focused on rule based systems. There was nothing found in the literature review that specifically discussed testing the unique aspects of alternative knowledge representation methods (i.e., inheritance properties in frame

based reasoning). Continued research is necessary for the successful development of expert systems employing specialized knowledge representation.

The next chapter proposes a testing plan for ESAAMS.

V. SOFTWARE TEST PLAN

A prototype test plan for validating and verifying the ESAAMS project is described within this chapter. Specific criteria which must be met by ESAAMS for the system to become operationally suitable are discussed. Finally, a user interface test for users of ESAAMS is described.

A. SCOPE

This Software Test Plan establishes a methodology for testing the Expert System Advisor for Aircraft Maintenance Scheduling (ESAAMS). ESAAMS is an expert system currently under development by students at the Naval Postgraduate School. Unique about this test plan is its actual development during the software life cycle's earliest stages.

B. ESAAMS BACKGROUND

The purpose of ESAAMS is to provide a prioritized listing of aircraft maintenance discrepancies given a wide range of determining factors and constraints. These factors range from skilled personnel available to conduct maintenance, to available parts, hanger space (if required)/deck space, special tools and testing equipment, to criticality of the asset based on the readiness of the squadron.

Aircraft maintenance control operates in a dynamic, high intensity environment. Maintenance work priorities are made

several times daily, often reordering as circumstances dictate. These decisions are frequently made under extremely demanding and time sensitive conditions. ESAAMS will assist a maintenance control chief in the prioritization of repairs of all aircraft, both in planned and unscheduled maintenance. ESAAMS is not designed to replace the expert maintenance scheduler but rather serve as an advisor and to reduce the time necessary to perform this task. Rapid turnover of military personnel, changing policies of maintenance officers, the commanding officer and higher authority, preclude ESAAMS from ever becoming a stand alone system. As mentioned, ESAAMS is intended to be an advisor, a tool to assist a knowledgeable scheduler in making a rational decision by using information captured within the knowledge base. ESAAMS will also provide this advisory service to maintenance personnel familiar with aircraft maintenance scheduling, but not necessarily experts.

C. UNIQUE CHARACTERISTICS OF EXPERT SYSTEMS

Expert systems differ from conventional software in that they attempt to reproduce the mental procedures of a human expert while performing some task, as opposed to manipulating numbers as in conventional software. They are also able to deal with uncertainty and incomplete information. Due to its increased sophistication and ability to handle a different range of problems from conventional software, unique testing and evaluation procedures are required during the expert system software life cycle.

D. DESIGN OF EXPERT SYSTEMS

Testing and evaluation of expert systems, when done properly, starts early and continues throughout the entire design phase, user acceptance phase, and ultimately throughout the maintenance stage. System users are involved early on and are used throughout the testing cycle. In traditional system design, the developer begins with a written system requirement of what the system is supposed to accomplish, and designs accordingly. This set of standards is the basic functional foundation on which the developer depends.

The intent of the expert system developer is to reproduce the expert's mental procedures used when solving a problem. This includes the heuristics or "rules of thumb" developed through years of experience in a very narrow domain.

Most frequently the process is initiated by the knowledge engineer interviewing the expert. The expert's knowledge (heuristics) are translated step by step into a set of procedures used to solve a problem. A major complicating factor is that the experts may not be completely aware of why they do what they do. They have a difficult time translating the complete decision process of the human expert's knowledge into a set of procedures completely describing the process. Error is almost a certainty. The knowledge engineer and expert together analyze the procedures for correctness. After the expert has determined the translation is accurate, the

knowledge engineer inputs the set of procedures into a knowledge base as a set of rules or knowledge.

The initial prototype is now ready for testing. The expert system's built in syntax and rule checker first check for syntax errors and possible insertion errors. Then the system is ready for design testing, whereby the system's results are verified by the expert. Once test results are deemed acceptable, design work begins on the next iterative prototype.

Prototype testing and development continue until the system designer determines that the final prototype is fully tested in accordance with the specifications document.

Operational testing begins with users testing the prototype in the actual working environment. Testing continues until the designer, expert, and users are satisfied that the expert system meets their expectations and system specifications. The expert system is then ready for actual deployment.

The next section lists specifications which must be met if ESAAMS is to be considered successful.

E. CRITERIA

The following specific criteria must be met by ESAAMS for the system to be operationally suitable:

- ESAAMS must be capable of scheduling various quantities of aircraft (at least eight).
- ESAAMS must recommend the correct scheduling priorities with an accuracy acceptable by the expert.

- ESAAMS must recommend a scheduling process which can be performed by personnel having only a basic knowledge of maintenance scheduling procedures.

- ESAAMS must provide clear instructions. A self explanatory reasoning facility for leading personnel through scheduling procedures must be available.

- ESAAMS must complete a scheduling recommendation more quickly than would the user. Normally 15 minutes or less.

After it has been determined what ESAAMS is expected to accomplish, development of a testing plan defining all testing stages commences.

F. TESTING STAGES

Meister [Ref. 16:p. iii] categorizes testing of expert systems into three basic stages:

1. Initial design testing stage
2. Developmental testing stage
3. Operational testing stage

I have added a fourth stage, described as the:

4. Maintainability testing stage

These stages will be applied to ESAAMS for incorporation into the test plan.

1. Initial Design Testing Stage

The initial design testing stage consists of two parts: testing the knowledge base with the expert and debugging the software.

a. Knowledge Base Testing

Initial steps in the design of ESAAMS include the gathering of known data for the knowledge base, and

interviewing the maintenance scheduler (Maintenance Master Chief) for development of the heuristics used to manipulate information in the data store (inference engine). Recording the experts thought process is difficult, and must be reviewed thoroughly for accuracy.

Testing of known facts is the first step. Facts in the knowledge base must be verified through publications, local procedures, or any combination thereof. Examples of questions include:

- When a specific aircraft part is required, does supply normally carry it in stock?
- Is special test equipment required to perform a maintenance procedure?
- Does the aircraft require a hangar and/or electrical power for the repair work?

The second, and most difficult part of the evaluation, occurs when interviewing the expert for determining which heuristics are used to manipulate data in the knowledge base. It is not uncommon for the knowledge engineer to experience difficulties in expressing these rules of thumb accurately. A question that must be answered during evaluation is: Does the heuristic truthfully represent the process actually employed by the expert to manipulate the data store? [Ref. 16:p. 5]

- Why did the maintenance scheduler pick one aircraft with a longer fix time over the "quick fix" aircraft?
- Why did the maintenance scheduler prioritize the daily work schedule in this particular order?

- Why did the maintenance scheduler decide to take parts from the "hangar queen", rather than going through supply for the parts?
- What decision making process does the maintenance scheduler go through when determining which aircraft to work on next?

Once the interview(s) are complete, the knowledge engineer translates the data into a symbolic form recognizable by the computer. Once translation has occurred, an evaluation test is performed by returning to the expert, presenting him with the revised procedures, and determining if the new procedures match what was documented in previous interviews. This type of testing may continue for several iterations until a satisfactory match is found between the expert and knowledge engineer.

Unfortunately, there are no objective measures available for determining levels of success when performing this evaluation. The maintenance scheduler alone must determine whether he feels there is an accurate translation of his heuristics into a set of rules.

b. Software Testing

Software testing of the knowledge base in Nexpert Object is conducted using the rule network, syntax checker, and breakpoints as described in Chapter IV. Thorough evaluation is required each time new rules are added for consistency and completeness. When the designer and expert, after thorough review, believe most errors have been removed from the ESAAMS knowledge base, developmental testing begins.

2. Developmental Testing Stage

Following initial testing, developmental testing uses the computer and screen to improve the design of the expert system. Once the software is in place and been checked for syntax, the program now requires that the designer present the new prototype with a problem to solve. Initially the designer should present the system with simplistic problems, those with objective answers within predefined parameters. Certain nonsensical or contradictory answers are likely to appear, especially as segments of the system are linked together. The intent in developmental testing is <u>not</u> to determine the degree of system efficiency, but rather to ensure that apparently reasonable responses are secured from the software (no self contradiction, no obviously nonsensical outputs). [Ref. 16:p. 7]

a. Performance Effectiveness Testing

Performance effectiveness determines just how well the system performs during initial prototype testing and for each prototype developed thereafter. Does the prototype produce reasonable if not entirely correct answers? Will it do what it was designed to do? During this phase of testing, an expert needs to be available for judging the accuracy of results.

ESAAMS prioritizes maintenance work schedules for squadron aircraft. Maintenance scheduling is partially subjective, whereby entirely correct answers may not be known

and opinions may vary between experts as to which are correct answers. The following steps are proposed for conducting performance effectiveness training during the initial and later prototype development of ESAAMS:

1. Ensure both the expert(s) and an intended user are available for testing. Additional experts may be available from other squadrons in the air wing.

2. Gather maintenance action forms (MAF's) from a five day period. These MAF's will provide a wide range of possible test cases.

3. For testing purposes of the initial prototype, five sets of objects will be tested:

- Aircraft downing discrepancy (Is the aircraft fully mission capable with the discrepancy?)
- Parts availability (Are parts readily available from supply, or must they be robbed?)
- Skilled personnel to conduct maintenance (Are personnel of the proper rating available to work on the aircraft?)
- Electrical power/air (Is electrical power/air a requirement to fix the discrepancy?)
- Hangar/Deck (Is hangar space/deck space necessary to work on the aircraft?)

b. Test Cases

(1) First Test Case. For the first test case, enter all values into ESAAMS as TRUE. Does ESAAMS recommend that the aircraft be scheduled for maintenance now? Does the expert agree with ESAAMS?

(2) Second Test Case. Enter all values as FALSE. How does ESAAMS handle this problem? When does it recommend

the aircraft be scheduled for maintenance? Does the expert agree with the solution?

(3) Third Test Case. Enter all values as UNKNOWN. How does ESAAMS handle this unique case? What type of response appears? Does ESAAMS lock up?

(4) Fourth Test Case. Select the first actual MAF. Input all values. Does ESAAMS recommend starting maintenance on that particular aircraft? What does the expert recommend? Follow the networking procedure within NEXPERT OBJECT (trace facility) of the ESAAMS prototype to determine what logic was used to arrive at the solution.

(5) Additional Testing. Use the Turing test. Have ESAAMS produce a set of recommended maintenance priorities. Compare these against the experts. Compare differences to determine an accuracy level. Then determine if the expert will accept the prioritized maintenance actions recommended by ESAAMS.

As development of ESAAMS continues through successive generations of prototypes, values will be attached to objects included in the knowledge base. For example, values for estimated time to fix maintenance discrepancies are entered. ESAAMS must be able to handle these values, producing correct results when they are introduced. Initially values within the normal domain of values are used. Sensible answers validated by the expert are necessary. Then values outside the prescribed domain are added to determine how

ESAAMS reacts. Finally, values at the "boundary edge" are tested. ESAAMS should also handle these values smoothly.

Continue testing the prototype throughout the entire range of MAF's, comparing ESAAMS results against actual maintenance actions taken. Does the expert feel the answers produced by ESAAMS are reliable? What does the expert feel is an acceptable range of differing recommendations from his own? For example, is he willing to accept a 75% figure of answers which match his own. Remember, the expert may not be accurate 100% of the time.

Throughout testing, the user should be involved early on. User acceptance will be critical to acceptance of ESAAMS as an advisory tool. ESAAMS must meet user needs on human factors and usability issues, as well as knowledge based issues. This topic will be discussed in the following section.

As development of ESAAMS proceeds, maintain a database within NEXPERT OBJECT of common cases that were previously tested. These same test cases are to be used as new knowledge is added to the system.

Check to see what differences have occurred since the new knowledge has been added. What new rules have fired? Is the maintenance priority still sensible? AS ESAAMS grows, this testing responsibility becomes an essential part of the prototyping process, and reduces the requirement for the expert to be present during all testing.

Testing should be conducted as often as possible. Each time a new object or variable is added to ESAAMS, thorough testing is to be conducted. The test should be run again to determine what effect the new value will have. The expert should validate the results for correctness. Using earlier, documented test cases will ensure a smoother transition between prototypes, while freeing up the expert.

c. Attribute Testing

Attribute testing prioritizes a listing of factors which determine how useful the system is in terms of certain characteristics. Sizemore [Ref. 17:p. 35] presents a listing of eleven software quality factors. These factors range from performance (efficiency, integrity, reliability, and usability), to design of the system (correctness, maintainability, and testability) to adaptability (flexibility, interoperability, portability, and reusability). These subfactors are described in Figure 3.

Reliability, the ability to perform with correct (or acceptable) and consistent results, is the key ingredient looked for in ESAAMS. All responses by ESAAMS must be acceptable by the expert in terms of scheduling priorities.

	PERFORMANCE	
EFFICIENCY	The ability of a software system to perform its required functions with minimum consumption of computer time and storage resources.	How well does it utilize resources?
INTEGRITY	The ability of a software system to control unauthorized access to or modification of system software or data.	How secure is it?
RELIABILITY	The ability of a software system to perform its required functions with correct and consistent results.	What confidence can be placed in what it does?
USABILITY	The ability of a software system to be easily learned and used.	How easy is it to use?
	DESIGN	
CORRECTNESS	The extent to which the software satisfies its specification and fulfills the user requirements.	How well does it conform to the requirements?
MAINTAINABILITY	The ability of a software system to be easily corrected when errors are discovered.	How easy is it to repair?
TESTABILITY	The ability of a software system to be easily and thoroughly tested.	How easy is it to verify its performance?

Figure 3
Software Quality Factors

ADAPTABILITY

FLEXIBILITY	The ability of a software system to be easily modified to meet new requirements.	How easy is it to change?
INTEROPER-ABILITY	The ability of a software system to effectively exchange information with other software systems.	How easy is it to interface with another system?
PORTABILITY	The ability of a software system to be easily modified to operate in more than one environment.	How easy is it to transport?
REUSABILITY	The ability of a software system or parts of a system to be used in multiple applications.	How easy is it to convert for use in another application?

Figure 3 (Continued)

Usability is important for user acceptance, and rates high on ESAAMS desirable features. Personnel must feel comfortable with ESAAMS for the system to gain wide acceptance. Does ESAAMS save the user time?

Correctness, or the extent to which ESAAMS fulfills the user requirements and expectations is necessary for acceptance. Does ESAAMS produce a usable schedule? Does the schedule make sense?

Flexibility and maintainability are important issues within ESAAMS. ESAAMS must have the ability for upgrade as requirements and knowledge change; e.g., the number

of aircraft in the squadron change, or priority changes as dictated by the commanding officer or maintenance officer.

Testability ranks high on a listing of desired features. What can be tested within ESAAMS as well as what cannot be tested, is important. In testing ESAAMS, the expert may not always be right. Additional experts are brought in to determine accuracy.

d. Acceptability Testing

Acceptability testing determines whether the end user will actually employ or use the system once it becomes available. The system must possess those attributes the user deems most important, and be able to deliver that attribute in a fashion that will enhance usage. The user evaluates the system attributes as described in the previous section, in a determination of what improvements or changes are necessary.

The following user interface questionnaire will give the designer a reasonable idea of how the user feels about ESAAMS, whether it meets their expectations, and the likelihood of further usage.

USER INTERFACE TEST

Instructions

In Section I., enter the time started when you turn on the computer. After being led through the procedures, enter time completed. After this process, advance to Section II.

I.

DATE: _____

TIME STARTED: _____

TIME COMPLETE: _____

II.

1. HAVE YOU EVER USED THE ESAAMS SYSTEM BEFORE?

   YES___NO___

2. A. HOW MANY HOURS OF TRAINING DID YOU RECEIVE ON ESAAMS BEFORE OPERATING THE SYSTEM? _____

   B. DO YOU FEEL THAT YOU HAD ADEQUATE TRAINING?

   YES___NO___

   C. HOW MUCH MORE TRAINING WOULD YOU HAVE LIKED?

   _____

   D. WHAT AREAS OF TRAINING NEED TO BE INCREASED?

   _____

   _____

   _____

   E. LIST ANY TRAINING NOT PROVIDED THAT SHOULD BE INCLUDED.

   _____

   _____

3. A. DID YOU READ THE ESAAMS USERS MANUAL (IF DEVELOPED)?

YES____NO____

B. IF YES, HOW WOULD YOU RATE THE MANUAL?

VERY GOOD_____ GOOD_____ POOR _____ VERY POOR _____

C. COULD YOU FIND WHAT YOU NEEDED TO KNOW TO USE THE SYSTEM? YES_____NO_____

D. HOW WOULD YOU IMPROVE THE MANUAL? _____

_____

4. A. HOW WOULD YOU RATE THE ESAAMS SOFTWARE ON EASE OF USE?

VERY EASY____EASY____DIFFICULT____VERY DIFFICULT_____

B. WHAT WOULD YOU LIKE TO SEE CHANGED, IF ANYTHING?

_____

5. HOW WOULD YOU RATE ESAAMS ON USER FRIENDLINESS I.E., LEADING YOU THROUGH THE PROCESS YOU SELECTED?

VERY GOOD_____ GOOD_____ POOR_____ VERY POOR_____

6. A. HOW WOULD YOU RATE ESAAMS ON LINE HELP FACILITY?

VERY GOOD____ GOOD_____ POOR_____ VERY POOR_____

B. WHAT ADDITIONAL ON LINE HELP FEATURES WOULD YOU RECOMMEND? _____

_____

_____

7. A. HOW WOULD YOU RATE ESAAMS PERFORMANCE REGARDING THE TIME TO MAKE A SCHEDULING DECISION?

VERY GOOD_____ GOOD_____ POOR_____ VERY POOR_____

B. WHAT ARE YOUR PERFORMANCE EXPECTATIONS WITH REGARD TO TIME IN MAKING A SCHEDULING DECISION? _____

_____

8. A. WHAT IS THE QUALITY OF ANSWERS PRODUCED BY ESAAMS?

VERY GOOD_____ GOOD_____ POOR_____ VERY POOR_____

B. WHAT ARE YOUR EXPECTATIONS FOR THE QUALITY (CORRECTNESS) OF ANSWERS PRODUCED BY ESAAMS?

_____

_____

_____

9. A. HOW CONFIDENT ARE YOU IN THE QUALITY OF ANSWERS PRODUCED BY ESAAMS?

VERY HIGH____HIGH____AVERAGE____LOW____VERY LOW____

10. A. HOW CONFUSED WERE YOU WHEN THE SOFTWARE WAS ASKING YOU TO INPUT OR SELECT SOMETHING?

NOT CONFUSED AT ALL___CONFUSED____VERY CONFUSED___

B. WHAT WAS CONFUSING?

_____

_____

11. A. HOW MUCH TROUBLE DID YOU HAVE SELECTING THE CORRECT KEYS TO PRESS TO DO WHAT YOU WANTED TO DO?

NONE AT ALL____ A LITTLE BIT____ QUITE A BIT___

B. IF YOU HAD TROUBLE, WHAT WERE YOU ATTEMPTING TO DO?

_____

_____

12. A. DID THE SOFTWARE FAIL AT ANY POINT? YES_____NO_____
(IF NO, SKIP TO 13.)

B. DESCRIBE THE FAILURE?

_____

_____

13. A. DID THE SYSTEM PROVIDE CORRECT AND TIMELY INFORMATION FOR YOU TO USE IN SCHEDULING THE AIRCRAFT FOR MAINTENANCE? YES____NO_____

B. AT ANY POINT DID YOU HAVE TO REQUEST HELP FROM YOUR SUPERVISOR?

YES_____NO_____ (IF NO, SKIP TO 14.)

C. WHAT ASSISTANCE DID YOU REQUEST OF HIM/HER? _____

_____

14. DID THE SYSTEM (IN YOUR OPINION) RECOMMEND THE PROPER SCHEDULING?

YES_____NO_____

15. ESAAMS IS A GREAT TIME SAVER OVER THE CURRENT SCHEDULING METHODOLOGY.

STRONGLY AGREE__AGREE___DISAGREE___STRONGLY DISAGREE___

IF YOU DISAGREE, PLEASE EXPLAIN: _____

_____

_____

16. WOULD YOU PREFER USING THIS SYSTEM OVER THE CURRENT METHOD? YES____NO____

17. WOULD YOU RECOMMEND THIS SOFTWARE FOR THE NAVY?

YES____NO____

18. OTHER COMMENTS: (PLEASE LIST ANY OTHER ITEM OF INTEREST)

_____

_____

_____

3. Operational Testing

Operational testing takes place when ESAAMS is considered complete by its developers and ready to be turned over to the users. Training on the system is the same as that to be given to the ultimate users.

Maintenance scheduling is to be performed by the users both with and without ESAAMS, for purposes of comparison. Differences, with ESAAMS hopefully documenting an increase in performance and efficiency, will significantly increase the value of the expert system.

Another comparative measure is to have the users of ESAAMS compare their performance to that of the expert. If expert status is attained by the user, or an approximation of expert status, ESAAMS will be validated as a useful tool.

a. Plan for Conducting Operational Testing

ESAAMS is being tested in an operational environment to determine if it can assist the user in reaching the same levels of expertise as the maintenance scheduler.

ESAAMS will be used to solve a variety of scheduling problems, all new and independent from previously used test cases. Results will be documented. The degree of "expertness" achieved by the user will be measured by the actual expert to determine the usefulness of ESAAMS.

Criteria for acceptance:

- ESAAMS must prioritize aircraft scheduled for maintenance at the same speed or faster than the user could do alone.

- ESAAMS must assist the user in producing a maintenance schedule deemed acceptable by the experts.

- All experts participating must agree on a common solution to a scheduling problem. System performance must meet standards set by the experts.

b. Acceptance of ESAAMS

ESAAMS will be considered acceptable when:

- The expert system equals or achieves the performance level of the expert.

- When left on line for an extended period of time, 48 hours, there are no system failures.

- When the user is able to follow the system prompts, and reach a solution, without added assistance from the system developers.

- When the users have complete confidence in the system and feel ESAAMS is a helpful tool (as annotated in questionnaires).

- Reasonable output is attained, given reasonable input.

4. Maintainability Testing Stage

Planned maintenance of ESAAMS is a difficult issue. Failure to address this topic may have dire consequences. Using obsolete information, or wrong information, will lose some hard won confidence that may be difficult to regain.

It is expected that the knowledge in ESAAMS will remain relatively stable. New systems added to the aircraft or new prioritization procedures put in place will affect this stability.

Recommended is that a "valid until date" be placed within ESAAMS on the opening screen, when ESAAMS is first brought on line. Beyond this date the system must be examined

for changing knowledge. A one year validation date seems appropriate.

ESAAMS is to be maintained by the Naval Aircraft Maintenance Office (NAMO). Additions, deletions, or modification of knowledge will only be accomplished by an expert system developer familiar with ESAAMS. It is hoped that as expert systems become more visible throughout the Navy, personnel experienced in their development will become available.

Testing is accomplished in exactly the same manner as was accomplished during development. Test cases will be run, with results verified against an experts judgment. Approval of multiple test cases will validate change to ESAAMS.

E. DOCUMENTATION

A progress notebook is to be maintained throughout the entire life cycle of ESAAMS, with testing results meticulously recorded by the development team. Results are to be compared against previous test case results, to determine where errors may have occurred. The progress notebook will be stored at NAMO, and used for updates of ESAAMS.

F. SUMMARY

This chapter has presented a formal approach to testing an ESAAMS prototype. Specific criteria have been annotated to assist in the development of ESAAMS. Four testing stages are described: the initial design testing stage, the developmental

testing stage, the operational testing stage, and the maintainability testing stage.

A user interface test to be completed by users of ESAAMS has been included. Finally, a short discussion on maintenance and documentation of ESAAMS concludes the chapter.

The next chapter will focus on recommendations and conclusions from this thesis for a successful development of the ESAAMS project.

VI. RECOMMENDATIONS AND CONCLUSIONS

The size and complexity of the decision domain to be incorporated into ESAAMS presents tremendous challenges to the developers. Below are recommendations and conclusions based on this specific research effort.

A. RECOMMENDATIONS

This study has focused on the validation, verification, and testing of expert systems, with a proposed test plan for ESAAMS included. Much work remains before ESAAMS will be available for testing in an operational squadron. The following lessons learned are a result of this thesis.

1. Requirements Document

Development of a sound, well thought out requirements document is the next and most crucial step for the successful deployment of ESAAMS. Future designers must pinpoint exactly what is expected of ESAAMS and how it will schedule maintenance. Specifications for ESAAMS must be developed. The requirements need to be thoroughly reviewed by both developers and users for accuracy and attainability. Iterative knowledge base developments and requirements refinement will follow. Some specifications are inclusive in the ESAAMS test plan, but further refinement is necessary. Figure 4 [Ref. 12:p. 17] lists a requirements generation process useful for ESAAMS.

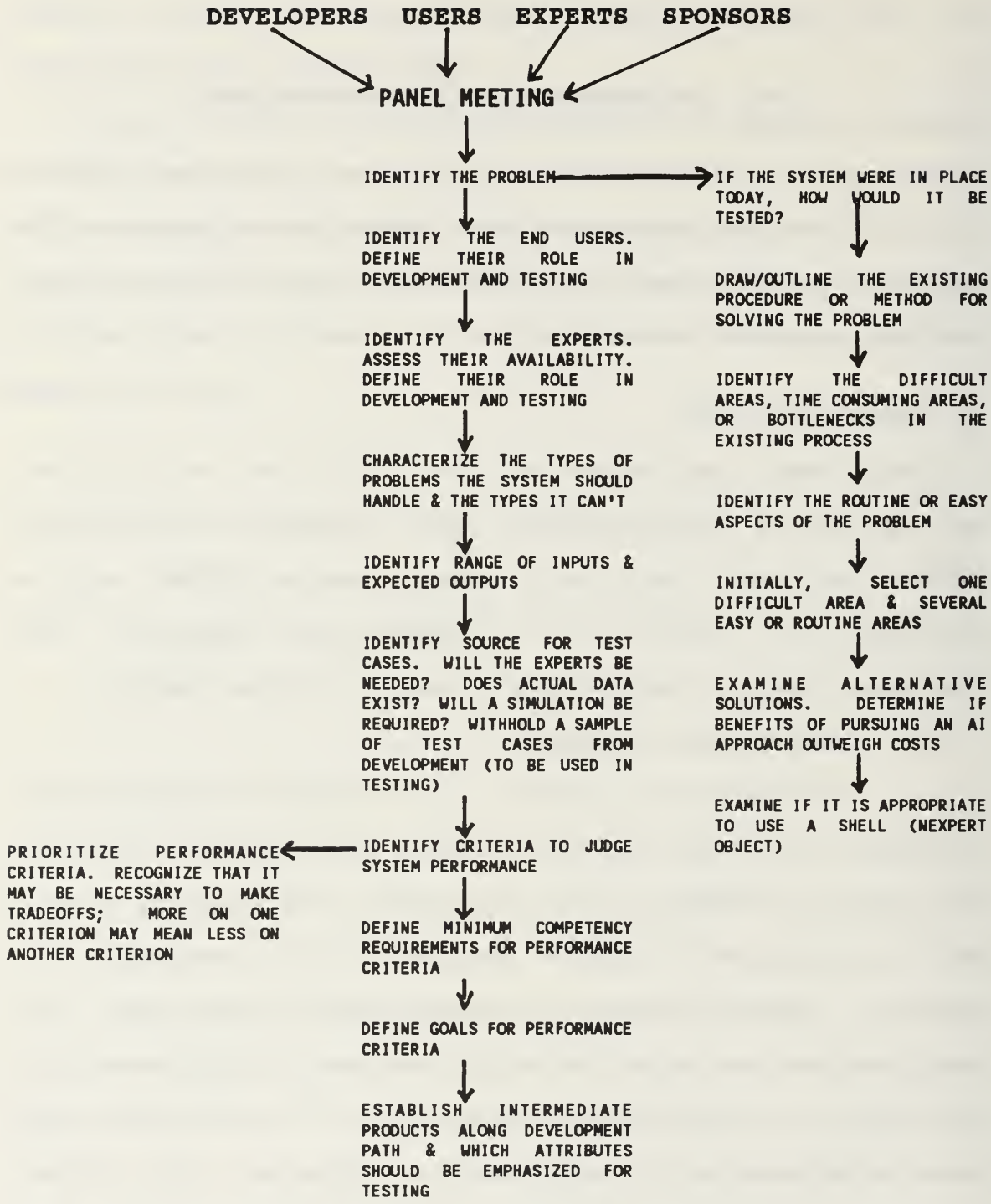


Figure 4
Requirements Generation Process

2. Ensure the Expert is Available

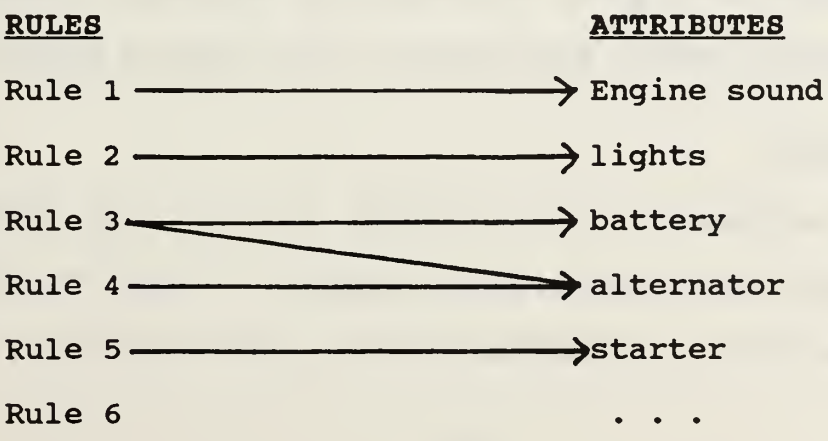
Initially, the Maintenance Master Chief from the VFA squadron selected for development of ESAAMs was always available. However, due to world crisis, the squadron unexpectedly deployed, forcing system designers to rely on their own expertise in aircraft maintenance for development and testing of ESAAMS.

3. Field Test Early and Often

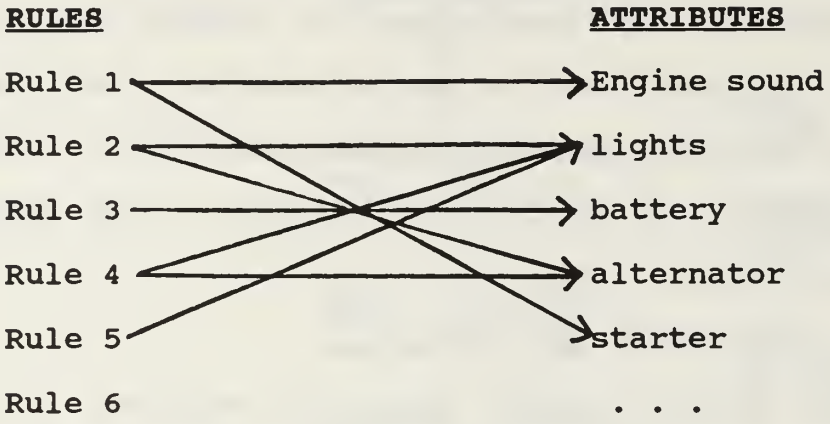
Expert system testing cannot be confined to those tests performed by the developers alone. Testing must sample the outside population, using ESAAMS under real world conditions.

4. Keep Conclusions Simple

When writing the rules for the knowledge base, keep them as simple and concise as possible. Firing of a rule should lead to a conclusion which instantiates only one attribute. Pedersen [Ref. 18:p. 24] provides an example of a well structured relationship among rules and attributes. One or more rules are responsible for concluding any one attribute:



Conversely, a poorly structured relationship is defined as follows:



Conceptualization of how different rules interact in a network and which conclusions are triggered by rule firings becomes excessively difficult as more rules are added to the knowledge base. Therefore, keep conclusions simple.

5. Establishment of a Maintenance Plan

The question of who will maintain ESAAMS is an important one. Usage of the expert system after knowledge obsolescence will cause users to lose confidence in the tool. How quickly the knowledge changes and what new knowledge is desirable are features necessary for a sound maintenance plan. A well thought out plan of who will maintain ESAAMS is necessary before actual deployment of the expert system.

B. CONCLUSIONS

Further refinement of the testing plan will be necessary once the requirements document is complete. Aviation officers attending the Naval Postgraduate School, and especially those

with aviation maintenance experience, should be encouraged to conduct follow on thesis research in the development of ESAAMS. However, this is not to suggest excluding those students interested in designing an expert system with no prior background in aviation.

Additionally, academia will benefit by maintaining an interest in expert systems development. New shells are becoming ever more available, technology is advancing at a rapid speed, and development of expert systems is expected to rise exponentially over the next decade. Development of ESAAMS will provide valuable insight for incorporation of a leading edge technology into today's complex military environment.

ESAAMS is an ambitious project which has never been attempted before. Once accomplished, it is expected that future expert systems of a significant nature will be realized.

LIST OF REFERENCES

1. McCaffrey, M. J., The Feasibility of Implementing An Expert System For Aircraft Maintenance Scheduling With The Naval Aviation Logistics Command Management Information System (NALCOMIS), Thesis, Naval Postgraduate School, September 1985.

2. Walker, T. C., and Miller, R. C., Expert Systems Handbook, The Fairmont Press, 1990.

3. Royce, W. W., "Managing the Development of Large Software Systems: Concepts & Techniques," Proceedings WESCON, August 1970.

4. Boehm, B. W., "A Spiral Model of Software Development and Enhancement," Computer, Vol. 21, No. 5, May 1988, pp. 61-72.

5. Kameny, I., Khan, U., Paul, J., and Taylor, D., Guide for the Management of Expert Systems Development, The Rand Corporation, July 1989.

6. AIRMICS, "Expert Systems Evaluation Methodology," July 1989.

7. Greene, C., and Keyes, M., "Verification and Validation of Expert Systems," IEEE, 1987.

8. Bonnet, A., Haton, J-P, and Truong-Ngoc, J. M., Expert Systems: Principle and Practice, Prentice Hall, 1988.

9. Prerau, D. S., Developing and Managing Expert Systems, Addison-Wesley Publishing Co., Inc., 1990.

10. Marcot, B., "Testing Your Knowledge Base," AI Expert, July - August 1987.

11. O'Keefe, R., "Validating Expert System Performance," IEEE Expert, Winter 1987.

12. Constantine, M., and Ulivla, J., "Knowledge Based Systems In The Army: The State Of the Practice and Lessons Learned, With Implications For Testing," Decision Science Consortium, 1990.

13. Bielawski, L., and Lewand, R., Expert Systems Development; Building PC Based Applications, QED Information Science, Inc., 1988.

14. Ngugen, T., Perkins, W. A., Laffey, T., and Pecora, D., "Knowledge Base Verification," AI Expert, Summer 1987.

15. Kang, Y., and Bahill, T. A., "A Tool For Detecting Expert System Errors," AI Expert, February, 1990.

16. Meister, D. D., "Behavioral Test And Evaluation Of Expert Systems," Navy Personnel Research and Development Center, San Diego, 22 September 1987.

17. Sizemore, N. L., "Test Techniques for Knowledge-Based Systems," ITEA Journal, Vol. X1, No. 2, 1990, pp. 34-43.

18. Pedersen, K., Expert Systems Programming: Practical Techniques For Rule Based Systems, John Wiley & Sons, 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5000	2
3.	Prof. Martin J. McCaffrey (Code AS/Mf) Naval Postgraduate School Monterey, California 93943-5000	4
4.	Prof. Tung X. Bui (Code AS/Bd) Naval Postgraduate School Monterey, California 93943-5000	1
5.	Mr. Robert Harder STEEP-ET-S Ft. Huachuca, Arizona 85613-7110	1
6.	CDR Tom J. Hoskins (Code 37) Naval Postgraduate School Monterey, California 93943-5000	1
7.	LCDR Christian W. Andrieu VQ-4 NAS Patuxent River, Maryland 20670	2