Developing a Strategy for Expert System Verification and Validation

Sunro Lee and Robert M. O'Keefe

Abstract—**Research and practice has produced numerous methods for expert system verification and validation (V&V) that augment traditional software and systems approaches to V&V. These methods, however, are not associated with software development methodologies, as has been the case with conventional V&V, and there is evidence that this is a considerable stumbling block in expert system V&V. This paper shows how to develop a strategy for expert system V&V. A strategy for V&V has three components: 1) the criteria by which an expert system will be judged as valid, 2) a life cycle model that specifies what V&V can be done when, and 3) the constraints (and opportunities) imposed by the characteristics of the system being developed. Starting with a development methodology, we shown how to map V&V methods onto the software life cycle and then match V&V methods to system characteristics. We give an example of our approach to developing a V&V strategy in the context of an expert system development project.**

I. Introduction

NOW that expert systems (ES) have reached the stage where they are implemented and used in a wide variety of organizations, verification and validation (V&V) has become very important. In particular, systems where erroneous advice may lead to loss of life, loss of considerable amounts of money, or damage to expensive physical facilities have engendered a need for careful V&V.

Validation focuses on 'accurate performance' reflecting the needs of the user and the organization, and has been defined as building the 'right system' [38]. Verification, on the other hand, refers to building the 'system right,' often focusing on the implementation of specifications [38]. These definitions imply that validation is more result-oriented and is for ensuring right solutions with user satisfaction, whereas verification is process-oriented and is for achieving completeness of necessary specifications.

Within traditional software engineering (SE), V&V approaches have typically assumed clearly predefined requirements and computationally expected solutions [5]. Based on system requirements, verification has been conducted on the code level by static analysis of data flow and program structure [3], and validation is frequently conducted by dynamic analysis using test data. In the ES development environment, however, requirement specifications are often imprecise and rapidly changing [55], creating problems for specification-based verification. Further, expert knowledge may be incomplete, imprecise, and perhaps even lacking, with the result that even knowing when the ES is performing incorrectly may be difficult. Hollnagel [26] puts this very succinctly: "the paradox in applying ES is that we want them to do perfectly things we don't really understand."

Given these problems, researchers have recognized the importance of ES V&V, analyzing previous validation studies [41], providing frameworks and guidelines [12], [24], [38], [49], [51], and developing automated verification tools [13], [46]. Such research, however, has not been tightly coupled with software development methodologies, as has previously been the case with conventional V&V. A recent survey of ES developers [23] suggests that this is a considerable stumbling block when ES developers have to determine what V&V should be done.

This paper considers how to develop a strategy for ES V&V. A strategy has three components: 1) the criteria by which an ES will be judged as valid, 2) a life cycle model that specifies what V&V can be done when, and 3) the constraints (and opportunities) imposed by the characteristics of the ES being developed, which also have an impact on the selection of what and when. We do not develop a single strategy—in fact we argue that a single universally viable strategy does not exist and thus putting together a strategy for ES V&V is often a difficult task.

In the next section, we discuss typical criteria for ES V&V. This is followed by a brief review of recently developed methods for ES V&V. We then use a previously developed life cycle model [39] based upon Boehm's [9] spiral development model as a starting point for our explanation of V&V strategy, followed by an analysis of typical ES system characteristics. We then show how a strategy for a particular system can be generated from all this, and present the application of these ideas to a fairly large ES project.

II. Criteria for Verification and Validation

Traditionally, software quality has been the main concern for V&V [3], [8] and the so-called quality assurance approach tends to emphasize the verification of system reliability, i.e., the removal of programming 'bugs.' Since an ES influences decision-making, often augmenting or replacing human decision making in an organization, closer investigation of the impact of the system on the user and the organization is called

Manuscript received February 7, 1992; revised December 1, 1992 and May 29, 1992. Paper recommended by Associate Editor, Andriole. This work was partially supported by the UNDP and the Egyptian Government under Contract EGY/88/024.

S. Lee is with the Business Information Systems, School of Business and Management, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.

R. M. O'Keefe is with the Decision Sciences and Engineering Systems, School of Management, Rensselaer Polytechnic Institute, Troy, NY 12180-3590 USA.

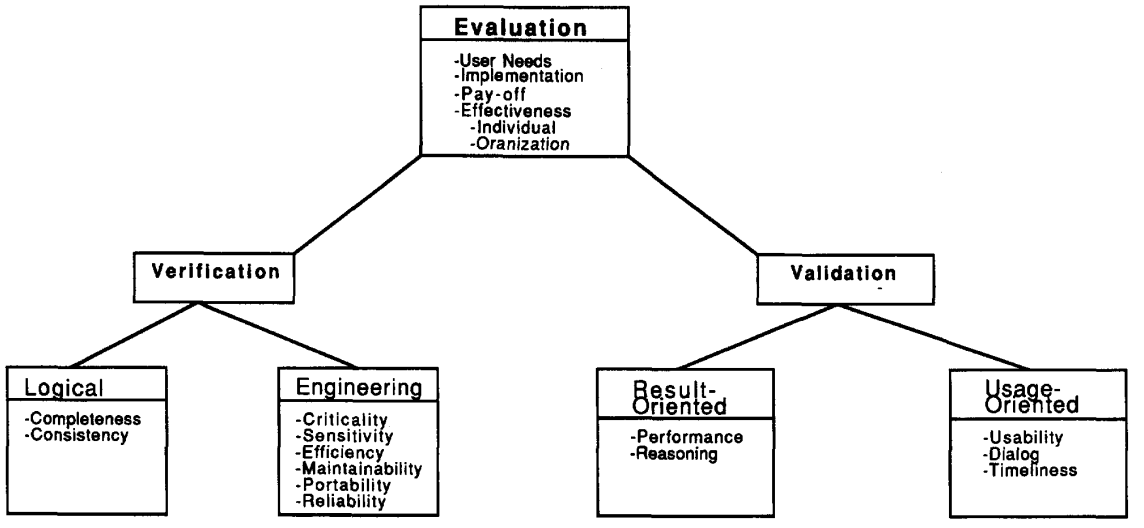

Fig. 1. Criteria for verification and validation.

for. This is particularly true for systems that work in a decision support mode [1], [37]. Integrating user and organizational perspectives with quality assurance attributes, we can develop criteria for ES V&V as shown in Fig. 1.

It is always necessary to perform logical verification on the knowledge base, which checks that the knowledge base is logically consistent and logically complete. When considering a rule base, for instance, an example of a consistency error is a redundant rule that can never fire in the course of inference; a possible attribute value that is never used is an example of a completeness error [36]. How a knowledge base is verified is a function of the technology employed—most developed methods relate to rules, since to this point rule-based systems have dominated practice, but many of the concepts are applicable to other knowledge representation (KR) schemes.

Complete verification must, as with any software, also address engineering issues, including considerations of criticality, sensitivity, efficiency, maintainability, and portability [21], and also reliability. Criticality analysis identifies the most important parts of the software and is useful in V&V since these can then be more thoroughly tested. Sensitivity refers to the ability of the software to gracefully degrade, a major problem with many ES. Efficiency, maintainability, and reliability are relevant to all software, knowledge-based or otherwise: the system must execute quickly enough for the given application, we must be able to easily update and alter the system and produce the same results under identical operating conditions. Portability is the ability to transfer the system to a different computer environment, which can be important, for example, when a prototype is converted to a delivery system for implementation.

Validation focuses both on results and usage. The system's performance can be compared with the expected performance, which itself can be prescribed or shown by comparison to human experts. A very common criterion is the acceptable level of performance [38] i.e., the level of performance that the ES is expected to exhibit by the sponsors and developers. Correct reasoning, obtaining the correct result by virtue of the correct line of reasoning, may also be important [41]. We suggest that results-oriented validation is normally a prerequisite for extensive usage-oriented validation—if an ES can not demonstrate reasonable performance, or at least the hope of future performance following further development, then usage issues are irrelevant.

Assuming performance can be achieved, then as the developed ES becomes more complete attention will shift to usage-oriented validation. So as to distinguish this from results-oriented validation, this is sometimes called assessment. For systems with a user, perceptions of features such as usability, timeliness of the advice and man-machine dialogue (for example control over explanation facilities) have been found to be important factors affecting system acceptance. Extensive lists of other possible criteria have been developed elsewhere (for example, [2], [48]), and any number, ranging from very few to very many, may be important in any single application. The relevant criteria will be very dependent upon the characteristics of the system and its intended use. Issues specific to ES, such as the quality of explanation facilities, may be present, but often criteria will represent features such as ease-of-use that are far removed from the underlying system technology and are important for any system.

From a managerial viewpoint, V&V are key parts of the overall evaluation process. At the start of a project, evaluation tries to determine the feasibility of and pay off from the proposed system [29]. In the final stages of a project, we are interested in specifying implementation, examining the effectiveness of a system, and ultimately estimating its 'pay off'.

III. Verification and Validation Methods: A Brief Review

We can broadly classify recently developed methods for V&V of ES into three categories: automated tools, analytical modeling, and human support. Automated tool building attempts to automate V&V by providing software which can

analyze systems written in a particular structure or shell, indicating potential errors in the implementation of the knowledge base (KB). Analytical modeling approaches attempt to quantify V&V criteria, for instance by formally measuring the performance of the ES. Human support relies on efforts by developers, users or experts to aid V&V.

A. Automated Tools

Automated tools address logical verification. They typically filter the KB through meta-constraints [13] or check its structure. This requires that an intermediate representation is used to represent the KB, such as a dependency chart [36], a decision table [16], a graph [46] or a Petri net [4], [34]. Automation can contribute to reducing the number of iterations in the system life cycle by discovering gaps in the expert's knowledge and errors in their reasoning processes during, rather than after, systems development [52].

The most extensive meta-constraint tool is the Expert Systems Validation Associate (EVA) [13], developed as a front-end ES verification shell at Lockheed. EVA interfaces with a standard ES shell, such as KEE, and facts and rules are translated into EVA format. Then, EVA verifies the ES using structure-checking algorithms and meta-knowledge. This meta-knowledge is expressed using EVA predicates, for example a predicate incompatible can be used to specify a combination of facts that should not co-exist in the KB. EVA then tries to prove that this situation can occur.

Examples of checking programs include the Expert System Checker (ESC) [16], the Rule Checking Program (RCP) [52], CHECK [36] and Validator [28]. These tools contain algorithms that check for known potential errors; for example, we can check for redundant rules by checking that the conditions in each rule can match with the actions of at least one other rule, and that the actions in a rule can match with the conditions in at least one other rule. COVER [45], [46] is an advance over these tools, employing a graph-based representation that allows for efficient checking over chains of inference, rather than comparing rules in pairs, as is done by the other tools. Petri net based approaches such as Liu and Dillon [34] hold out the promise of verifying dynamic and temporal relationships between rules, but converting a rule base into a Petri net is a nontrivial task.

Overall, automated tool approaches are limited to two-valued logic rule-based systems, make stringent assumptions about inference, and have yet to prove their applicability for KRs other than rules. Although they are designed to augment traditional SE approaches, it is unclear how they can be integrated with SE. Many developers, without recourse to an automated tool, take a standard SE approach to verification based upon manual inspection of the KB, dynamic testing with data, and inference tracing.

Engineering verification may also require SE methods beyond existing automated tools. A good example of this is testing for reliability where an ES can obtain its input in one of a number of different orders. (This commonly occurs with ES since the reasoning process will obtain data 'on the fly' in the course of inference.) It is necessary to show that the order in which the input is obtained does not affect the output, and this can be done by permutating the order, running the ES with each permutation, and comparing output.

B. Analytical Modeling

Analytical modeling methods attempt to provide quantitative methods to aid the V&V process. As a logical verification tool, an analytic model can provide insights into the verity of a KB. O'Leary and Kandelin [43] present models for investigating the weights in systems based on inference networks, with a view to determining the robustness and authenticity of the weights. Lehner [32] presents another testing procedure for use with inference networks. Langlotz *et al.* [31] used decision theory to verify the heuristics used in MYCIN, simulating the performance and interaction of the heuristics under differing input conditions.

Concerning engineering verification, measurements of execution times can be used to determine efficiency, and analytical rule grouping methods (such as shown in [27]) can help structure a rule-base so as to increase its maintainability. Again, analytical modeling should be complemented with traditional SE in order to achieve a full range of engineering verification. Miller [35] presents an approach for determining the most important types of error for a particular application, and suggests that, given available resources, testing efforts should be assigned to the possible errors which appear to be more critical. The subsequent testing can make use of any approaches, traditional or otherwise.

Analytical methods are primarily of value, however, for performance validation, particularly when comparing system results against expected performance. Validity can be established as a hypothesis test, with the acceptable level of performance determining the condition under which the hypothesis is accepted [38]. Simple statistical tests such as confidence intervals can be used in certain instances, but more normally nonparametric tests and consistency measures will be used (e.g., [25]). A more novel use of analytical methods is the construction of an analytical model, or use of a pre-existing model, as an alternative to a knowledge-based approach. Here the two methods can then be compared, with the performance of the analytical model giving insights into the performance and accuracy of the knowledge-based approach—a good example of this is the common approach of comparing an artificial neural network model against an equivalent quantitative model, such as a logistic regression or linear discriminant model [53]. Finally, an ES can be tested out using simulation, where the ES controls a simulation model and its performance is measured against acceptable limits (e.g., [22]). The validity of the ES is measured by evaluating the output of the simulation (or multiple simulation runs), perhaps by comparing it against expert performance or some notion of acceptable performance.

C. Human Support

Human support means testing of systems by developers, users or experts, and ranges from considering performance of the ES on a set of test cases early in its life, through to fielding

the system and monitoring it or obtaining feedback. Many ES use some form of human support for both result- and usage-oriented validation. No doubt the most popular result-oriented validation method is presently case testing, where historic cases are given to the system [11]. Evaluators can compare the ES results with predetermined acceptable performance or previous expert performance. Where relative performance can only be subjectively assessed, knowing where the results came from may produce human biases: evaluators are unfairly biased when comparing the performance of a limited ES with that of its more diversified human expert [20]. Hence a Turing test or blinded evaluation methodology is normally used, where third-party experts evaluate the results from the ES and other experts without knowledge of which is which [25].

Field testing can be conducted after completion of a prototype which displays acceptable performance on the entire problem. The prototype can be tested by the actual user group and errors or suggestions can be collected [15]. During field studies, sometimes an acceptable level of performance can be implicitly obtained from the user group, since it may become apparent what performance level users require to adequately perform tasks. For systems with a number of users, it may be possible to employ a control group or other quasi-experimental approach, where the system is implemented for some users, and performance is compared to a control group that do not have access to the system [1]. Field testing, then, can address both result- and usage-oriented validation, and the mixing of the two into a single test can be a problem. For this reason, often only systems that have adequately passed performance validation are fielded.

Human estimates of usage-oriented concerns, such as the usability, value and usefulness of the system, can be used to assess usage-oriented validity and the effectiveness of the system [48]. Usage data has to be collected, and this can be done by observation, questionnaire, or sometimes direct monitoring by the software. Reducing this data to an effective measure is very difficult, and requires that the comparative relevance of each usage criteria is measured. This can require that a standard multi-criteria structuring method, such as the Analytical Hierarchy Process (AHP), be used to obtain weights for each criteria. Liebowitz [33] shows how to do this using a simple AHP model, in the context of an ES developed at NASA. O'Keefe [37], Riedel and Pitz [48], and others have generalized this idea using additive multi-attribute models. In many instances, however, multi-criteria/attribute models should only be applied to usage-oriented validation *after* the ES achieves an acceptable level of performance, since performance and usability are not necessarily linked—the user may not be in a position to evaluate the performance of the system.

D. Summary

Table I summarizes and compares existing methods for V&V of ES with respect to important attributes. A more extensive review can be found in [40].

Each developed approach 1) focuses on a particular part of the evaluation process, 2) is appropriate at different times of the development process, and 3) has a number of potential problems. Automated tools only perform static logical verification, and are useful after a KB is formed and prior to validation. With analytical modeling, quantitative information can be obtained for engineering verification in general and result-oriented validation. Case testing can be done at any time, but cases are not always available. Use of a simulation model, a Turing test or a control group experiment are good formal validation procedures, but all require that the ES is well formed and generally verified, and all can be expensive. Field testing is cheap (assuming users of the system are not paid to use it or are not substantially distracted from other tasks), but is only possible for non-critical applications, and requires that the fielded system is properly monitored [15]. Based upon the above discussion, as shown in Fig. 2, the three broad categories of the V&V approach are mapped onto our criteria, showing that the different approaches support different V&V criteria.

What has happened to this point in time is that verification has dominated the V&V process, much as it dominates V&V in traditional SE. The move towards automated tools may make this worse: mechanical verification tends to finalize the whole process without any regard for results-or usage-validation. Even though the specifications are complete and consistent, the overall system might not meet user needs. Further, much validation of ES has been too informal and ad hoc (although exceptions to this can be found, e.g., [15], [47]), and has been conducted, without systematic verification, at the end of development.

Gaschnig *et al.* [20] explicitly mention the importance of formal V&V at an early stage of development. It is necessary to detect major deficiencies in performance, and problems with use of the system, when there is still time and money to do something about it. We believe that V&V are not separable, but rather complementary to each other under the overall evaluation of ES as depicted in Fig. 1. Therefore, it is important to fit V&V approaches into the overall ES development activity in order to broaden the leverage of V&V. What is needed is a strategy that features:

1) Tight coupling: close mapping of V&V to the particular ES development life cycle being used, in terms of both timing and focus.
2) Matching of V&V techniques with system characteristics: systematic matching of V&V methods with the characteristics of the ES.

The survey of Hamilton *et al.* [23] suggests that the lack of 1) and the inability of developers to do 2) are presently major recognized problems in ES V&V. As such, they require solution.

IV. Mapping of V&V to Development Stages

Several methodologies for software development are in use. Many of these have been found to be difficult to apply to ES, since the nature of ES development is generally different from that of traditional software:

1) There are often ambiguities in basic requirements, sometimes reflecting difficulties in knowledge acquisition (KA).

TABLE I
COMPARISON OF EXPERT SYSTEM VERIFICATON AND VALIDATION METHODS

	Examples of Methods	Approach	Focus	Timing	Potential Problems
Automated Tools	EVA [1]	Meta-constraints	Logical verification	After KB formulation	Context dependent, Static
	COVER [45]	Structure-checking algorithm	Logical verification	After KB formulation	Static
Analytical Modeling	Analysis of Heuristics [31]	Decision-theoretic algorithm	Logical validation of heuristics with uncertainty	After Knowldge acquisition	Inaccuracies and biases in assessing probability
	Statistical Methods [38]	Confidence intervals Consistency measures	Result-oriented validation of performance	After prototyping	Only supporting evidence
	Weight Analysis [43]	Bayesian inference	Logical verfication of oncertainty weight	After knowldge acquisition	Computational complexity
	Simulation [22]	Control of simulation model	Result-oriented validation of performance and Engineering verification	After mature prototyping	Requires full system development, expensive
	Alternative Model [53]	Comparison against other model	Result-oriented validation of performance	After mature prototyping	Interpretation of comparisons
Human Support	Case Testing [11]	Comparison of solutions from experts and systems	Result-oriented validation of performance	After prototyping	Case availability, Human biases
	Turing Testing [25]	Blind comparison of solutions by third party	Result-oriented validation	After mature prototyping	Case availability, Expert time
	Field Testing [15]	End-user feedback	Result- and usage-oriented validation	After mature prototyping	Only for non-critical applications
	Control Group [1]	Comparison between groups	Result- and usage-oriented validation	After partial implementation	Requires partial implementation
	Data Collection [2]	Data collection by observation or questionnaire	Usage-oriented validation	After partial implementation	Defining criteria

2) Heuristic knowledge can rarely be systematically delivered to knowledge engineers, who may themselves not be familiar with the problem domain. Hence what is 'doable' may not be evident early on.
3) The changing environment in which the ES operates tends to make a KB unstable and requires it to be refined as necessary.

Whereas each of these problems can be true for other types of information system [6], they are almost always present with ES development. Hence the problems need to be addressed in each project as they arise, rather than ignored or avoided by using a particular development methodology.

It has been suggested that such instabilities can be properly covered by applying evolutionary and spiral development life cycle models to ES [50]. The spiral model is a risk-driven approach to the software process rather than a specification or code-driven process [8], with the distinctive feature that it includes explicit stages for risk analysis and forces the system designers to express the criteria for entering and leaving each stage. Each level of software specification is followed by a verification step and the preparation of plans for the next cycle. Determination of entering and leaving criteria can help the developers set clear objectives for the system and help the validator to evaluate the system without bias.

A. An Integrative Model

To explain the issues and problems involved in mapping V&V methods onto a life cycle, we present here an integrative model that maps many of the generally applicable V&V methods onto the spiral model [39]. Fig. 3 shows a modified spiral model for ES development.

Our intention is not to present this as *the* way to develop an ES, but to use it as an example in what follows. Given the discussion below, it will be realized that a similar mapping could be done for any iterative development methodology, such as general prototyping approaches (e.g., [42]) or more

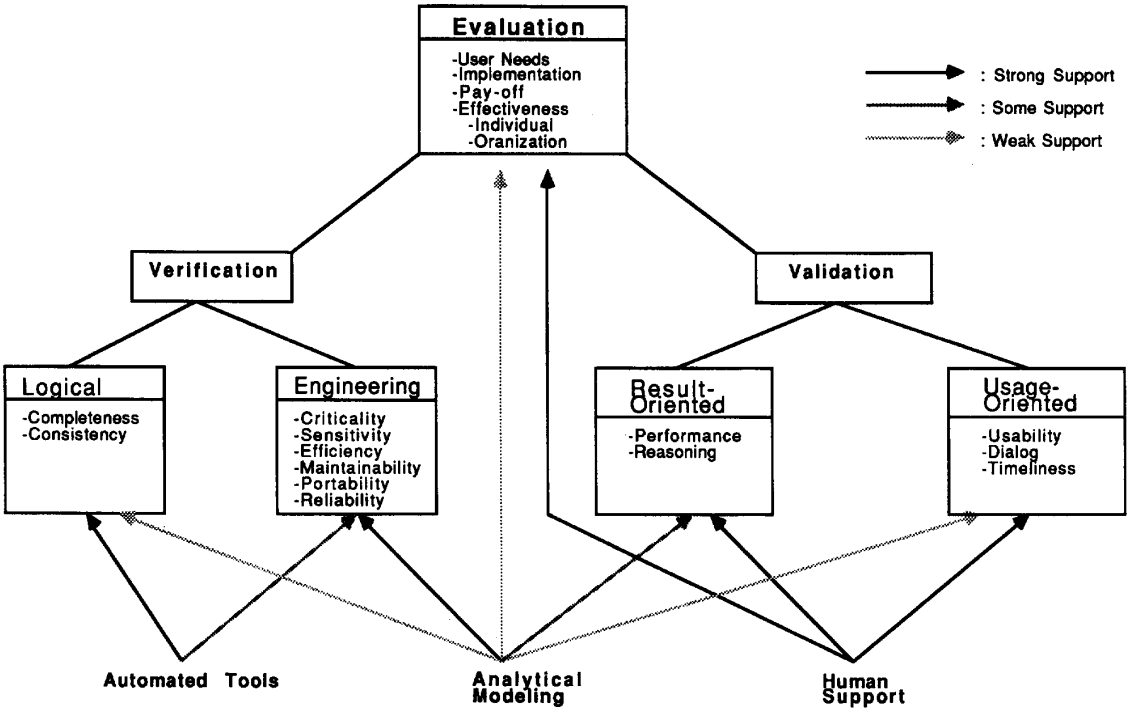

Fig. 2. Verification and validation criteria and current support.

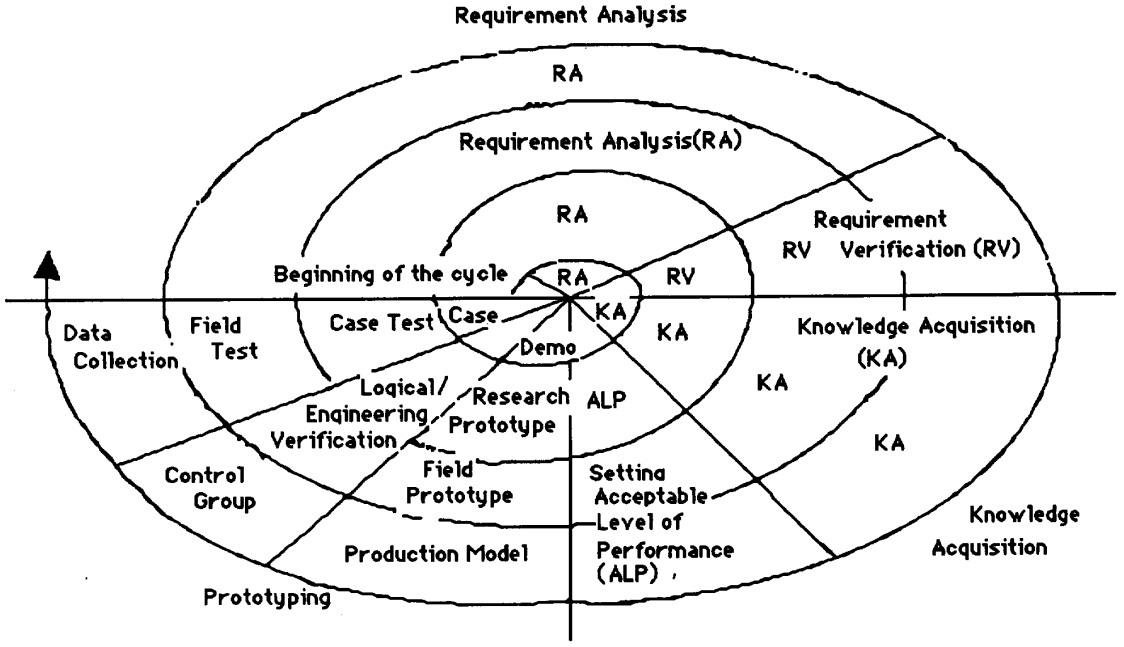

Fig. 3. The integrative model for expert system development.

specific ES methodologies, such as Weitzel and Kerschberg's KBSDLC [55], [56].

Note that, in the integrative model, the acceptable level of performance is the driving criterion. This is often defined in initial requirements, but is typically refined and expanded after KA. It can be operationalized in many ways, for example, performance on test cases, performance in a Turing test, performance in the field, etc. In later development spirals, however,

usage-oriented criteria may become important depending on the type of ES being developed.

In this model, a cycle in the ES development life cycle mainly consists of four stages: requirement analysis (RA), the KA process, prototyping, and implementation and maintenance. V&V processes have been embedded in each stage, but the focus of V&V will not be the same at each stage. We therefore consider each stage in turn.

B. Requirement Analysis (RA)

The fit between the system and the organization should be addressed from the beginning of ES development. The system must increase the performance of the users and the organization, and hence the bottom line is "Will the system add value?" [37]. The right problem and potential users should be addressed and the potential impacts of the ES on the organization should be investigated. The ES approach to solution of a certain task should have a high payoff in environments characterized by scarce human expertise or diminishing resources of expertise, and the task itself should be appropriate to the use of ES.

RA, the generation of requirements, will follow to define system functionalities. The initial RA for ES is primarily related to questions about what kind of problems can be solved under which environments. First, the specific problem for which the ES is being designed should be matched with proper ES building environments. Second, the problem needs to be decomposed into manageable sub-problems, upon which knowledge engineers can organize knowledge. Explanation and consultation processes should be analyzed depending on the user group, for example, naive users who have little domain knowledge will require detailed reasoning exposure and more flexible interfaces, whereas superusers who possess more understanding of domain knowledge need more abstract explanation and perhaps command driven interfaces

C. Knowledge Acquisition (KA)

KA is the process of gathering domain knowledge, typically from human experts, and transforming it to a certain KR scheme. Verification in KA can be considered a major task in the iterative development of ES—hopefully, a knowledge engineer can understand the boundaries of the available knowledge and subsequently the acceptable level of performance can be stated for the purpose of system validation.

Boose [10] suggests that knowledge engineers prepare diagrams representing a summary of the problem solution process and give it to the experts for verification. The diagram can be an influence diagram, activity graphs, decision tables, or classification hierarchies. This visual interaction can enhance mutual understanding of the problems and lead the domain expert to more structured reasoning. Verification facilities in an automated KA tool can provide real time verification of expert knowledge [7], and many automated KA tools now provide some method of automated verification, frequently a mechanism for checking for inconsistency or conflicts in the acquired knowledge.

Although considered separately here, RA and KA are increasingly being subsumed into a methodology that covers elements of both. A particularly good example is the KADS methodology [57], developed in Europe under the ESPRIT project, which provides methods and modeling tools to support the collection and analysis of knowledge. Knowledge is structured into a four-layer model of expertise. The lowest domain level consists of basic facts, concepts and relationships; the inference layer describes problem solving processes in a declarative manner; the task layer provides procedural descriptions of tasks that can be performed by using parts of the inference layer. The highest strategy layer is mean to provide models of complex problem solving and relate them to the environment, but this layer is often not considered by knowledge engineers.

D. Prototypes

Prototype systems can be constructed following some KA. Waterman [54] explains the evolution of ES with evolving prototypes, and describes three types of prototypes:

1) Demonstration or laboratory prototype: the system solves a portion of the problem, demonstrating ES capabilities and available development methodologies. This is sometimes called 'proof of concept'.
2) Research or mature prototype: the system shows credible performance on the entire problem but needs more testing and revision.
3) Field prototype: the system shows reliable performance with extensive testing in the user environment.

Incremental prototypes can serve as a source of requirements and enhance developers' understanding of the system's objectives and users' expectations. Such understanding may contribute to clarifying requirements and the acceptable level of performance in the next cycle. Each prototype, however, should be formally validated to gain the advantages of prototyping. As reviewed in the previous section, several V&V methods can be used: each method has pros and cons depending upon problem characteristics and the stage of development, and we therefore need to choose proper methods at each stage of development.

At the early stage of a demonstration prototype, mechanical verification using an automated tool can play an important role in detecting errors and preventing the cascading effect of early design faults on the evolving prototype. As a prototype matures, however, the focus of V&V will shift from logical verification to result-oriented validation.

The difficulties in collecting cases which cover a well defined problem boundary make case testing less meaningful at the early stage of development. Therefore, case testing becomes more appropriate after building a mature prototype which can handle at least a subset of complete cases. When good quality cases can be collected or generated, a subsequent Turing test can be very useful, but the cost of staging it and the problem of analyzing the results mean that it is often a 'one off' test. Thus it is typically used when a prototype is fully mature, perhaps ready for fielding.

Field testing can draw feedback from end-users, as previously discussed. As a prototype evolves to a production model and is fully institutionalized, a control group experiment [1] can be conducted for comparing system effectiveness and performance. Data collection can be performed to provide data for usage-oriented validation.

E. Implementation and Maintenance

After satisfactory development, one of two things can happen to an ES prototype. In some instances, it will serve as the specification for a system to be implemented and will be recoded in a conventional programming language. Software engineers will use conventional V&V to test the system, now possible due to the clear definition of the system embodied in the prototype. (Thus, somewhat paradoxically, engineering verification may only take place after validation when the system is recoded.) More normally the prototype will evolve into a production system and be implemented, and hence proper engineering of the KB, such as modularization [27], will be a precondition for cost effective maintenance.

When operational, the impacts of the system need to be monitored and conformance with requirements checked. As users' understanding increases, however, and users become more proficient in using the system, perceptions of the system change. Therefore, system usage must be properly documented for evaluating the utility of the current system and incorporating any new design requirements. Maintenance requires additional V&V, which Adrion *et al.* [3] term 'regression testing'. Old test cases should be run in order to detect the contradictions between the old and new knowledge base, and ensure the robustness of the system. In many instances, however, previous cases will not be appropriate since the boundaries of the KB has changed.

V. Matching V&V to System Characteristics

Whatever the development strategy used to develop the system, the characteristics of the ES being developed will also affect the choice of V&V methods. Quite obviously, for example, a small system in a non-critical domain developed for a single user will normally have less stringent V&V requirements than a large embedded system to be used in real time by numerous users. Here, we match V&V to various characteristics, namely KR, knowledge type, problem type, criticality, usage, functionality, interactivity and user profile.

A. Knowledge Representation

ES knowledge can be classified into two categories: declarative and procedural. Declarative knowledge is the facts and heuristics represented in the KB, whereas procedural knowledge is the strategies and procedures used in problem solving [14]. When using a simple rule-based ES shell, for example, ideally the rules should be expressed declaratively so that their ordering does not matter. Normally, however, developers order the rules so as to achieve a desired procedural flow in the system and may create so called meta-rules that determine problem solving procedures. Present automated tools assume only declarative knowledge, and make stringent assumptions about inference and procedural actions. It is thus likely that considerable amounts of procedural knowledge will require that the ES employs dynamic testing, testing the system under operating conditions. This is because static testing of knowledge may not be able to predict the interaction between declarative and procedural knowledge when the system actually executes.

B. Knowledge Type

The way in which knowledge is structured can also be a factor in selecting proper V&V techniques. ES typically differentiate between two types of knowledge: surface and deep. Surface systems typically utilize rules, with inference being performed by simple pattern matching. In a deep system, complete physical structures can be represented in the form of a mathematical model or a causal network.

Automated tools may not be effectively applied to V&V of deep systems, since the methods are specifically designed for surface systems. Similarly, case testing and Turing tests are well matched with surface knowledge, since it is easier to view the ES as providing correct or incorrect results with some justification for the result. Contrast this with, for example, neural networks, where it may be unclear why the system is not correctly performing over certain cases.

For deep models, a simulation model which can provide examples of causality against which a causal model can be compared is often very useful. Comparison against shallow modeling methods can indicate the value of the deeper approach, for instance, the previously mentioned tendency to validate neural network models against equivalent regression models.

C. Problem Type

Many authors, for instance Clancy [14], make a considerable distinction between analysis and synthesis. Analysis problems, such as classification, involve moving from data and facts to a correct answer or conclusion. Alternatively, synthesis problems, such as planning, involve combining diverse knowledge to produce a feasible plan or design. Usually, analysis problems employ declarative shallow knowledge, as typified by many ES shells, whereas synthesis problems require deeper knowledge that is both declarative and procedural [19].

With synthesis problems, it is less likely that the results of a test case can be easily evaluated as 'correct' or otherwise. This, in fact, may only be clear when the plan or design is implemented. Further, if the plan is correct in the sense that it is feasible, it still may not be the best possible plan. Hence case testing may not be possible, and in many instances only field testing will fully determine the performance validity of the ES.

The use of an alternative model can be extremely important for synthesis problems, since comparison against solutions from a quantitative model can sometimes determine the validity of the ES. The quantitative model may be only applicable to very bounded problems, but an ES that can produce the same results may be able to scale up to problems beyond the range of the quantitative model. For an example of this, see Ow and

Smith [44] on a comparison between the knowledge-based scheduling system OPIS and a simple heuristic scheduling rule. Alternatively, the quantitative solution may simply be unimplementable due to efficiency problems, and hence an ES approach is being developed. For an example of this, see Duchessi and O'Keefe [18] on where a knowledge-based production planning system is compared to the equivalent integer linear programming model.

D. Criticality

Perhaps the characteristic most often mentioned by researchers in ES V&V is the criticality of a domain. In critical situations, where loss of life or large sums of money can result directly from poor ES performance, case and Turing tests are unlikely to be stringent enough and field testing is just not possible. Complete enumeration of performance using simulation may be necessary, albeit expensive, and automated tools may be important final checks on completeness. It is thus no surprise that most work in building automated tools has been done in highly critical areas such as aerospace, military applications and medicine.

For non-critical systems, field testing will often be viable. If the efficacy of solutions from the ES can be easily determined by the users of the ES, for example, expert configuration systems where the resulting configuration simply will not work, then the ES can be allowed to fail and each failure can trigger alteration or further development of the ES. Where field testing is not viable, for whatever reason, case and Turing tests allow for formal validation.

E. Usage

Where usage of a system is optional, usage-oriented validity can not be fully assessed until the system is in use, since using a prototype under laboratory conditions can be very different from using the system in the field. Hence field testing, if possible, is the preferred method of human support. Following this, usage-oriented validation via some method of data collection is recommended.

When usage is enforced, due to outright replacement of existing methods or managerial decree, it will normally be necessary for V&V to be performed earlier in the development process. Usage-oriented validation may still be relevant, but this may have to be considered earlier in development, perhaps during prototyping. This is because if users do not understand the implemented system, or its performance is poor, enforced usage will magnify the resulting problems.

F. Functionality

A characteristic similar to usage is functionality: whether the ES replaces the existing method of problem solving or supports it. Supporting ES are liable to be optional, and hence the above comments apply. Further, a control group approach can be very useful for evaluating the performance of users supported by the new ES against previous methods without support. Alternatively, for a replacement system, V&V is liable to be more stringent, and again extensive testing using simulation may be appropriate.

G. Interactivity

Increasingly, ES are embedded in hardware or other software, and often such systems are critical. Here, human support V&V may not be very useful, and stringent verification using automated tools and simulation is important. Due to interfacing with other systems such as databases, input–output systems and the like, traditional V&V and quality assurance may well dominate specific V&V of the ES component. For stand-alone ES, such as simple ES developed using shells, V&V will be far easier since it does not have to address system integration.

H. User Profile

Finally, who is the final user of the system will have a considerable effect on the choice of V&V. Where the user is a novice and particularly where use of the system is enforced, the user is not in a position to help determine the validity of the ES. Hence a Turing test is liable to be a preferred approach since it can determine the acceptability of the ES by experts. Alternatively, expert users, liable to have ES that work in a support mode, may be in a position to judge performance and thus field testing will probably be preferred. In order to achieve usage-oriented validity, empirical results on designing user interfaces can provide guidance. For example, novice users have been shown to perform better in terms of speed and accuracy when interfacing with concrete questions rather than an abstract representation [30].

Whatever system characteristics are considered, and the list developed above is by no means exhaustive, it is evident that some characteristics (or sets of characteristics) will dominate V&V choices. This can be seen in present approaches to V&V. For example, the V&V of critical, enforced, embedded ES, as found in the military and aerospace, is by necessity formal and mechanical [35]. This is in contrast to non-critical, optional, support systems, where verification need not be so stringent, and whose validation is far more akin to that for Decision Support Systems [2]. When a system falls into a broad category, such as these two examples, choice of V&V methods and development of a V&V strategy will be far easier.

System characteristics will, of course, also have a considerable impact on the development life-cycle used. Thus the two can not be considered totally separately, and in fact will often provide similar reasons for choice of V&V methods. For instance, in the same sense that critical systems suggest a need for formal and mechanical verification, critical systems are likely to be developed using a formal development process.

VI. Developing a Strategy for V&V

It is evident, then, that to properly verify and validate an ES we must have a strategy that combines a specification of V&V criteria (based upon any requirements criteria), a careful appraisal of where V&V methods can be employed in the development life cycle, and due consideration of the characteristics of the ES. We need to *map* methods onto the life cycle, and match methods to system characteristics.

Given this, we suggest the following way to develop a strategy:

1) Choose and specify the criteria for V&V: This should be done in as much detail as possible. For instance, if we choose acceptable level of performance as our major criterion, then we may specify it in terms of the performance of the ES on a set of predefined cases or a simulation model. Although this can be changed 'down the line,' and for some systems may not become clear until some KA is performed, having a clear as possible goal for V&V will greatly aid development.

For many ES, level of performance will be the dominating criterion. However, particularly where usage is optional, usage-oriented criteria should also be developed early on, even if how to measure for usage-oriented validity may not be immediately apparent.

2) Develop a list of suitable V&V methods given the intended characteristics of the ES and the identified criteria: Using the eight attributes discussed above, and possibly others, it should be relatively easy to generate suitable V&V methods. This may need to be done in tandem with 1), since certain criteria are best measured using certain methods (e.g., acceptable level of performance almost always requires some human support methods or the use of a simulation model so that the 'level' can be measured).

3) Map the V&V methods onto the life cycle being used: Specify where the methods will be used and when. Take account of the timing requirements of V&V methods (see Table I) and the focus of each method. This mapping can be revised, but revision should not mean frequently postponing V&V until the later stages of the life cycle (where issues of cost, timing and resource availability will come into play).

This may appear to be little more than common sense. Our experience, however, supports the findings of Hamilton *et al.* [23] that few developers enter an ES project with any preconceived ideas about how to do V&V.

Assuming we go through the above process, how do we determine that the resulting strategy is a good one? Obviously, if the resulting ES is valid and usable, then the strategy for V&V can be considered successful; however, developers will want to have confidence in the strategy before employing it. At the least, the V&V strategy will itself have to undergo face validity: Is it 'doable' ? Does it make unreasonable demands on the budget or project timing? Will it generate enough credibility in the system with users and project sponsors ?

Example: As an example of these ideas in action, we present the V&V efforts associated with a system under development in Cairo, Egypt, to support agricultural advisors in their planning and analysis of cucumber production. Here, we will call the system ESCPM, or Expert System for Cucumber Production Management, although in the project different prototypes have been given different names.

Cucumber is a major crop in Egypt, mainly grown under cover (so-called 'plastic tunnels') to protect the crop from the harsh sun. Planning decisions have to be made regarding planting, fertilization and irrigation, and then diagnosis and treatment is necessary if the plants develop any problems, especially diseases. ESCPM is designed to capture the expertise of agricultural experts in these various areas, and deliver it to advisors in the field on a personal computer. The delivered system will be comprised of different subsystems for the various planning and diagnosis tasks, and each subsystem is presently at different stages of development.

The criteria for ESCPM were initially specified in the project plan. Acceptable level of performance was formally stated in terms of the number of queries to advisors that the system would help with, and its congruence with the advise that actual experts would give. For usage-oriented criteria, requirements were far less precise and open to interpretation: the main concern of the development team was the ability to deliver a multilingual version of a system, usable in English or Arabic.

Regarding ES characteristics, the system as originally envisaged was a declarative, shallow analysis system. Crop and environmental data would be input into a mainly rule-based system that would give certain recommendations. Further, the characteristics of the application were considered as:

1) Non-critical, since most bad decisions can be subsequently rectified. (There is, however, a risk associated with incorrect advice, which will normally manifest itself as lower crop yield.)
2) Stand-alone, since all necessary data must be available to the ES on the portable PC.
3) Optional, since field advisors can not be forced to use it, with the system playing a supporting role in consultations. Further, field advisors are normally novices, unable to judge the validity of the ES recommendations, and requiring the interface to be in Arabic.

It was initially determined that case and field testing would be the main V&V methods. Various cases were available, and others could be generated with expert help. These could be used throughout development to validate the performance level of the system. Since the system is non-critical, and designed to work in a support mode, fielding a mature prototype to a user group was considered an important step prior to full implementation. As discussed below, a Turing test has been subsequently added to the development process.

The project does not use an explicit life cycle model, but it is employing an iterative methodology with three distinct stages—RA/KA, prototyping, and testing—similar to the integrative model presented earlier. Table II shows the V&V methods that are presently mapped onto three prototypes, of which (at time of writing) the first is finished, the second is in progress, and the third is planned.

Prototype 1 is essentially a laboratory prototype designed to prove the concept, and it mainly implements the diagnosis and treatment of disease subsystem. Since no automated tool was available to aid verification, logical verification was performed by hand. This involved manual static analysis of each rule by comparing it to other rules, and also dynamic analysis using artificial test data. This uncovered some problems with subsumed rules (where a less specific rule can always fire in preference to a more specific rule), such that the system might suggest a particular treatment and additionally a more specific instance of that treatment, but this did not affect the performance of the system since it would report the more specific treatment.

TABLE II
BASIC ELEMENTS OF V&V STRATEGY
FOR EXAMPLE, ES

Prototype	V&V methods employed or planned
1. proof of concept using diagnosis subsystem	Logical verification
	Case testing-performance on 69 cases compared against performance of 5 experts
2. full system, including task planning	Logical verification
	Boundary testing
	Regression testing on prototype 1 cases
	Turing test-evaluation by 2 experts of 3 experts and system performance on each sub-system
3. field system	Prototype 2 acts as specification-implemenation checked for exact conformance
	User questionnaire, infield observation

Validation was done by comparing performance of the system on 69 cases against performance by five experts; of these, 28 cases really tested the diagnosis and treatment knowledge. Each case was run through the system and every expert. For diagnosis and treatment, the experts had to complete a form where they marked the diseases they thought would be present and the treatment they would recommend. Those appearing on the form represented those modeled in the ES; additional treatments could be written on the form. Comparison between the system and the five experts on these 28 cases proved difficult, since a) the experts did not agree, and b) different diagnoses could actually lead to similar treatments, and thus 'agreement' could be measured in two different ways: agreement on diagnosis, or agreement on treatment. Whereas agreement on diagnosis was very mixed between all experts and the system, when subsequent treatments were considered the system and the key expert on plant disorders only disagreed in seven of the 28 cases. This was the lowest observable number of disagreements, i.e., disagreement rates between the system and the other four experts, and the key expert and the other four experts, were always greater than seven.

Following experience with prototype 1, three things happened. First, it was determined that a better methodology for doing KA and designing the system were necessary, and hence the KADS methodology [57] (briefly discussed earlier) has been adopted. Second, validation of prototype 1 suggested that the use of uncertainty measures were problematic, and that these should be dropped. The system's internal use of measures on a continuous scale had led to problems in ascertaining exactly what each measure should be; given that the system converted final uncertainty measures to a discrete scale (low, medium and high), it appeared more appropriate to reason with these measures internally. Third, due to the problems in comparing experts and the system, it was decided that the validation of prototype 2 would use a Turing test where two of the five experts would evaluate the performance of the other three and the system in a blinded evaluation. The use of two experts is necessary to give complete 'top expert' coverage in all areas, and also provides for some comparison across evaluators in important areas.

Further, RA and KA revealed that ESCPM needed to be a deeper system involving some synthesis tasks. This is mainly in the area of irrigation, where a detailed plan for water usage is necessary. This also necessitated a move towards a Turing test approach, since it is difficult to evaluate a plan, since very different irrigation plans may be equally acceptable as they result in the same distribution of water to the crop.

Prototype 3, which will be fielded, will simply be a programmed version of prototype 2, freeing the implementation from use of a proprietary shell. Hence, it is envisaged that prototype 2 will undergo formal SE testing, for example boundary testing, and that prototype 3 will then be tested for exact conformance to prototype 2. This is an example of using a prototype as a specification for the implemented system.

What is missing from V&V to the present time is user involvement, and V&V focusing on usage-oriented criteria. Ideally, prototype 2 should be exposed to users, but this presents difficulties due to the different geographic location of the project and the field advisors. Experts, some of which have previously advised in the field, have been used as surrogates. It is planned that the system will be fielded to a limited user group, and that this will be accompanied by a questionnaire on use and some infield observation. Interestingly, however, because usage criteria have not been explicitly developed, designing this part of the V&V process is proving difficult. It is very unlikely that a formal multi-criteria approach will be usable.

Although in the middle of things, this example shows quite clearly a number of key points. First, the importance of criteria: these drive the V&V effort, and the more explicit they are the easier the V&V process will be. Second, characteristics help determine the selection of methods, and these must be mapped onto the development cycle. They can not be left until the end of development, but have to be integrated into the whole development process.

Third, a V&V strategy will change over time. This may happen due to some combination of three reasons: 1) the criteria by which the system is to be judged change, 2) the characteristics of the system change, or 3) difficulties in employing one V&V method will necessitate a change to another approach. Both 2) and 3) are evident in the ESCPM example, and there is no guarantee that 1) will not occur once the system is fielded. Hence, a V&V strategy can not be laid out at the front of a project and then blindly followed.

The strategy itself must be dynamic, and may need changing following each cycle in the development process.

VII. Conclusion

V&V is vital to the success of ES, yet guidelines on both ES development (for example, [55]) and management of ES projects (for example, [17]) give scant regard to V&V. We propose that developers of ES should have a strategy that puts the criteria for V&V 'up front', and maps V&V methods onto the development life cycle being used after selection following a *match* against system characteristics. The result is a plan that shows what V&V can be done when.

There is no one single strategy that is viable for all ES, as hinted elsewhere (e.g., [47], [45]). Even systems with similar characteristics will likely have different strategies if the life cycles are different. What we present is an attempt at a meta-strategy, a way to help work out what a particular V&V strategy will be. Further, as shown in our example project, this strategy will change as development proceeds, and hence although doing V&V early is generally a good thing, over commitment to specific methods is not.

It was mentioned above that V&V strategies for common types of systems that exhibit key characteristics (such as critical, enforced, embedded ES) are emerging. This is a sensible short-term approach to the problem of developing a strategy for V&V, particularly in organizations where each ES is of a single type. The hidden danger is that a specific strategy will be lifted whole to another project, even though the structure and characteristics of the new project make the single developed strategy inappropriate. There is no substitute for a general appreciation of how to do V&V.

Acknowledgment

We are grateful for the comments provided by Martha Grabowski, Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute. The project producing the ESCPM system is headed by Dr. Ahmed Rafea of the American University in Cairo, and the second author is very grateful for being provided with the opportunity to plan the V&V for the project.

References

[1] L. Adelman, "Experiments, quasi-experiments, and case studies: a review of empirical methods for evaluating decision support systems," *IEEE Trans. Syst, Man, and Cybern.*, vol. 21, no. 2, pp. 293–301, 1991.

[2] ——, *Evaluating Decision Support and Expert Systems.* New York: Wiley, 1992.

[3] W. R. Adrion, M. A. Branstad, and J. Cherniavsky, "Validation, verification, and testing of computer software," *Computing Surveys,* vol. 14, no. 2, pp. 159–192, 1982.

[4] R. Agarwal and M. Tanniru, "A Petri-net approach for verifying the integrity of production systems," *Int. J. Man-Machine Studies,* vol. 36, no. 3, pp. 447–468, 1992.

[5] S. J. Andriole, *Software Validation, Verification, Testing and Documentation.* Princeton, NJ: Petrocelli Books, 1986.

[6] D. E. Avison and A. T. Wood-Harper, "Information systems development research: an exploration of ideas in practice," *Comput. J.,* vol. 34, no. 2, pp. 98–112, 1991.

[7] I. Benbasat and J. S. Dhaliwal, "A framework for the validation of knowledge acquisition," *Knowledge Acquisition,* vol. 1, no. 1, pp. 215–233, 1989.

[8] B. W. Boehm, "Verifying and validating software requirements and design specification," *IEEE Software,* pp. 75–88, January 1984.

[9] B. W. Boehm, "A spiral model of software development and enhancement," *IEEE Comput.*, pp. 61–72, May 1988.

[10] J. H. Boose, *Expertise Transfer for Expert System Design.* New York: Elsevier, 1986.

[11] B. G. Buchanan and E. H. Shortliffe, *Rule-based Expert Systems: The MYCIN Experiments of The Heuristic Programming Project.* Reading, MA: Addison-Wesley, 1984.

[12] G. Castore, "Validation and verification for knowledge-based control systems," in *Proc. 1st Annual Workshop on Space Operations Automation and Robotics,* Houston, TX, Aug. 1987, pp. 197–202.

[13] C. L. Chang, J. B. Combs, and R. A. Stachowitz, "A report on the expert systems validation associate (EVA)," *Expert Systems with Applications,* vol. 1, no. 3, pp. 217–230, 1990.

[14] W. Clancy, "Heuristic classification," in *Knowledge-Based Problem-Solving,* J. Kowalik (Ed.), Englewood Cliffs, NJ: Prentice-Hall, 1986, pp. 1–67.

[15] T. Cochran and B. Hutchins, "Testing, verifying and releasing an expert system: the case history of mentor," in *Proc. the Third IEEE Conf. on AI Applications,* 1987, pp. 163–167.

[16] B. Cragun and H. Steudel, "A decision-table-based processor for checking completeness and consistency in rule-based expert systems," *Int. J. Man-Machine Studies,* vol. 26, no. 5, pp. 633–648, 1987.

[17] J. M. Cupello and D. J. Mishelevich, "Managing prototype knowledge/expert system projects," *Comm. ACM,* vol. 31, no. 5, pp. 534–541, 1988.

[18] P. Duchessi and R. M. O'Keefe, "A knowledge-based approach to production planning," *J. of the Operational Research Society,* vol. 41, no. 5, pp. 377–390, 1990.

[19] C. Garg-Janardan and G. Salvendy, "A conceptual framework for knowledge elicitation," *Int. J. Man-Machine Studies,* vol. 26, no. 4, pp. 521–531, 1987.

[20] J. Gaschnig, P. Klahr, H. Pople, E. Shortliffe, and A. Terry, "Evaluation of expert systems: issues and case studies," in *Building Expert Systems,* F. Hayes-Roth, D. A. Waterman and D. B. Lenat (Eds.), Reading, MA: Addison-Wesley, 1983, pp. 241–280.

[21] C. J. R. Green and M. M. Keyes, "Verification and validation of expert systems," in *Proc. IEEE Knowledge-Based Engineering &Expert Systems,* IEEE, Piscataway, NJ, pp. 38–43, 1987.

[22] D. L. Hall and D. T. Heinze, "The use of simulation techniques for expert system test and evaluation," *ISA Transactions,* vol. 28, no. 1, pp. 19–22, 1989.

[23] D. Hamilton, K. Kelley and C. Culbert, "State-of-the-practice in knowledge-based system verification and validation," *Expert Systems with Applications,* vol. 3, no. 4, pp. 403–410, 1991.

[24] P. R. Harrison, "Testing and evaluation of knowledge-based systems," in *Structuring Expert Systems,* J. Liebowitz and D. A. DeSalvo (Eds.), Englewood Cliffs, NJ: Yourdon Press, 1989, pp. 303–329.

[25] D. H. Hickam, E. H. Shortliffe, M. B. Bishchoff, A. C. Scott, and C. D. Jacobs, "The treatment advice of computer based cancer chemotherapy protocol advisor," *Annals of Internal Medicine,* vol. 103, no. 6, pp. 928–936, 1985.

[26] E. Hollnagel, "Issues in the reliability of expert systems," in *The Reliability of Expert Systems,* Hollnagel, E. (Ed.), Chichester, UK: Ellis Horwood, 1989, pp. 169–209.

[27] R. J. K. Jacob and J. N. Froscher, "A software engineering methodology for rule-based systems," *IEEE Trans. Knowledge and Data Engineering,* vol. 2, no. 2, pp. 173–189, 1990.

[28] M. J. Jafar and A. T. Bahill, "Validator, a tool for verifying and validating personal computer based expert systems," in *Operations Research and Artificial Intelligence: The Integration of Problem Solving Strategies,* D. E. Brown and C. C. White (Eds.), , Boston, MA: Kluwer, 1990.

[29] P. W. Keen, "Value analysis: justifying Decision Support Systems" *MIS Quarterly,* vol. 5, no. 1, pp. 1–15, 1981.

[30] D. M. Lamberti and S. L. Newsome, "Presenting abstract versus concrete information in expert systems: What is the impact on user performance?," *Int. J. Man-Machine Studies,* vol. 31, no. 7, pp. 27–45, 1989.

[31] C. P. Langlotz, E. H. Shortliffe, and L. M. Fagan, "Using decision theory to justify heuristics," in *Proc. AAAI-86,*, Menlo Park, CA: AAAI, 1986, pp. 215–219.

[32] P. Lehner, "Towards an empirical approach to evaluating the knowledge base of an expert system," *IEEE Trans. Sys. Man Cybern.,* vol. 19, no. 3, pp. 659–662, 1989.

[33] J. Liebowitz, "Useful approach for evaluating expert systems," *Expert Systems,* vol. 3, no. 2, pp. 86–96, 1986.

[34] N. K. Liu and T. Dillon, "An approach towards the verification of expert systems using numerical Petri nets," *Int. J. of Intelligent Systems,* vol. 6, pp. 255–276, 1991.
[35] L. A. Miller, "Dynamic testing of knowledge base using the heuristic testing approach," *Expert Systems With Applications,* vol. 1, no. 3, pp. 249–269, 1990.
[36] T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. Pecora, "Checking an expert systems knowledge base for consistency and completeness," in *Proc. of IJCAI-85,* AAAI, Menlo Park, CA, 1985, pp. 374–378.
[37] R. M. O'Keefe, "The evaluation of decision-aiding systems: guidelines and methods," *Inform. Management,* vol. 17, pp. 217–226, 1989.
[38] R. M. O'Keefe, O. Balci and E. P. Smith, "Validating expert system performance," *IEEE Expert,* vol. 2, no. 4, pp. 81–89, 1987.
[39] R. M. O'Keefe and S. Lee, "An integrative model of expert system verification and validation," *Expert Systems with Applications,* vol. 1, no. 3, pp. 231–236, 1990.
[40] R. M. O'Keefe and D. E. O'Leary, "Expert system verification and validation: a survey and tutorial," *Artificial Intelligence Review,* vol. 7, no. 1, pp. 3–42, 1993.
[41] D. E. O'Leary, "Validation of expert systems with applications to auditing and accounting expert systems," *Decision Sci.,* vol. 18, no. 3, pp. 468–486, 1987.
[42] D. E. O'Leary, "Expert system prototyping as a research tool," in *Applied Expert Systems,* E. Turbin and P. Watkins (eds.), Amsterdam: North-Holland, 1988, pp. 17–32.
[43] D. E. O'Leary and N. Kandelin, "Validating the weights in rule-based expert systems," *Int. J. Expert Systems,* vol. 1, no. 3, pp. 253–279, 1988.
[44] P. Ow and S. Smith, "Two design principles for knowledge-based systems," *Decision Sci.,* vol. 18, no. 3, pp. 430–447, 1987.
[45] A. D. Preece, "Towards a methodology for evaluating expert systems," *Expert Systems,* vol. 7, no. 4, pp. 215–223, 1990.
[46] A. D. Preece, R. Shinghal, and A. Batarekh, "Verifying expert systems: a logical framework and a practical tool," *Expert Systems With Applications,* vol. 5, pp. 421–436, 1992.
[47] A. E. Radwan, M. Goul, T. J. O'Leary, and K. Moffitt, "A verification approach for knowledge-based systems," *Transportation Research-A,* vol. 23A, no. 4, pp. 287–300, 1989.
[48] S. L. Riedel and G. F. Pitz, "Utilization-oriented evluation of decision support systems," *IEEE Trans. Sys. Man Cybern.,* vol. SMC-16, pp. 980–996, 1986.
[49] J. Rushby, "Quality measures and assurance for ai software," NASA Contract Report 4187, NASA, Washington, DC, 1988.
[50] R. A. Stachowitz and J. B. Combs, "Validation of expert systems," in *Proc. of the 20th Hawaii Int. Conf. on Sys. Sci., IEEE,* Piscataway, NJ, 1987, pp. 686–695.
[51] C. Y. Suen, P. D. Grogono, and R. Shingahl, "Verifying, validating and measuring the performance of expert systems," *Expert Systems With Applications,* vol. 1, pp. 93–102, 1990.
[52] M. Suwa, A. C. Scott, and E. H. Shortliffe, "An approach to verifying completeness and consistency in a rule-based expert system," *AI Mag.,* pp. 16–21, 1982.
[53] K. Y. Tam and M. Y. Kiang, "Managerial applications of neural networks: the case of bank failure predictions," *Management Sci.,* vol. 38, no. 7, pp. 926–947, 1992.
[54] D. A. Waterman, *A Guide To Expert Systems.* Reading, MA: Addison-Wesley, 1986.
[55] J. R. Weitzel and L. Kerschberg, "Developing knowledge-based systems: Reorganizing the system development life cycle," *Comm. ACM,* vol. 32, no. 4, pp. 482–488, 1989.
[56] J. R. Weitzel and L. Kerschberg, "A system development methodology for knowledge-based systems," *IEEE Trans. Sys. Man Cybern.,* vol. 19, pp. 598–605, 1989.
[57] B. Wielinga, A. Th. Schreiber, and J. A. Breuker, "KADS: A modelling approach to knowledge engineering," *Knowledge Acquisition,* vol. 4, no. 1, pp. 5–53, 1992.

Sunro Lee received the Ph.D. degree from Renssalaer Polytechnic Institute, Troy, NY. He is currently a Lecturer in the department of Business Information Systems at the Hong Kong University of Science and technology, Hong Kong.

His current research interests include evaluation and maintenance of knowledge-based systems and human factors in software engineering.

Dr. Lee is a member of DSI, TIMS, and ACM.

Robert O'Keefe received the Ph.D. degree from the University of Southampton, England. He is currently an Associate Professor in the Department of Decision Sciences and Engineering Systems at Renssalaer Polytechnic Institute, Troy, NY.

His current research interests include the verification, validation, and implementation of knowledge-based systems and their impact on organizations and the application of simulation and knowledge-based methods in manufacturing.

Dr. O'Keefe is a member of the following editorial boards:*International Journal of Intelligent Systems in Accounting, Finance, and Management* and*International Journal of Applied Expert Systems*; he is an Associate Editor of*ACM Transactions on Modeling and Computer Simulation.* He is a member of AAAI, TIMS, ORS, SCS, ACM, and BCS.