Artificial Intelligence Review 7, 3–42, 1993.

Expert system verification and validation: a survey and tutorial

ROBERT M. O'KEEFE

Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, U.S.A.

and

DANIEL E. O'LEARY

Graduate School of Business, University of Southern California, Los Angeles, CA 90089-1421, U.S.A.

Abstract. Assuring the quality of an expert system is critical. A poor quality system may make costly errors resulting in considerable damage to the user or owner of the system, such as financial loss or human suffering. Hence verification and validation, methods and techniques aimed at ensuring quality, are fundamentally important.

This paper surveys the issues, methods and techniques for verifying and validating expert systems. Approaches to defining the quality of a system are discussed, drawing upon work in both computing and the model building disciplines, which leads to definitions of verification and validation and the associated concepts of credibility, assessment and evaluation. An approach to verification based upon the detection of anomalies is presented, and related to the concepts of consistency, completeness, correctness and redundancy. Automated tools for expert system verification are reviewed.

Considerable attention is then given to the issues in structuring the validation process, particularly the establishment of the criteria by which the system is judged, the need to maintain objectivity, and the concept of reliability. This is followed by a review of validation methods for validating both the components of a system and the system as a whole, and includes examples of some useful statistical methods. Management of the verification and validation process is then considered, and it is seen that the location of methods for verification and validation in the development life-cycle is of prime importance.

ACM Categories and Subject Descriptors: D.2.4 [Software Engineering]: Program Verification — Validation; D.2.5 [Software Engineering]: Testing and Debugging; 1.2 [Artificial Intelligence]: Applications and Expert Systems; K.6.1 [Management of Computers and Information Systems]: Project and People Management — Life Cycle.

Key Words: expert systems, knowledge-based systems, verification, validation, testing, evaluation, credibility, assessment, development, life cycle, statistics.

1. INTRODUCTION

Expert systems incorporate human expertise in computer programs to allow these programs to perform tasks normally requiring a human expert. More formally, an expert system (ES) has been defined as 'a computing system capable of representing and reasoning about some knowledge-rich domain with a view to solving problems and giving advice' (Jackson 1986) and 'a computer model of expert human reasoning, reaching the same conclusions the expert would reach if faced with a comparable problem' (Weiss and Kulikowski 1984). Examples of developed and implemented systems include R1/XCON (Bachant and McDermott 1983), which configures VAX computers for Digital, Exper-TAX (Shpilberg and Graham 1989), developed by Coopers & Lybrand to give advice on corporate tax planning, ONCOCIN (Langlotz and Shortliffe 1983), which helps doctors determine appropriate treatments for chemotherapy patients, and CLASS (Duchessi *et al.* 1988), a system that supports commercial loan decisions in a bank. Reviews of other systems can be found in Waterman (1986) and Ernst (1988).

Showing that an ES is in some sense 'correct' is a critical task. An incorrect system may make costly errors, or may not perform up to expectations. In either case the decisions generated by the system may be inappropriate or wrong, and if relied upon, considerable damage such as financial loss or human suffering may result to the user or owner of the system. For example, expert medical diagnosis systems and income tax systems have encountered implementation difficulties due to concerns over liability of the system's diagnoses.

It is sometimes suggested that due to their declarative nature and use of high level programming tools ESs are easy to verify and validate. As discussed by Fox (1990), this is only marginally true. A small rule-based program, for example, may be easier to verify that a large FORTRAN program, but large ESs encounter the same verification problems encountered by any large software project, and, as will be discussed, validation can actually be more problematic than with much traditional software.

The purpose of this paper is twofold: (1) to survey and integrate the literature on verifying and validating ESs, and (2) provide a tutorial introduction to the verification and validation of ES. Accordingly, this paper will discuss definitions for verifying and validating ESs, and discuss some of the primary approaches. Throughout, a fairly liberal interpretation is made of the term 'expert system'; we do not limit ourselves to rule-based systems based upon identifiable human expertise. However, as many existing ES fall into this category, a lot of what we discuss has been developed for exactly these type of systems.

2. VERIFICATION, VALIDATION AND SYSTEM QUALITY

2.1. *A hierarchic view of system quality*

Verification is defined by Adrion *et al.* (1982) as 'the demonstration of the consistency, completeness and correctness of the software'; as noted by O'Keefe *et al.* (1987), 'verification means building the system right'. Verification is aimed at eliminating errors in the system, and making sure that it corresponds to the specification. Adrion et al. (1982) indicate that 'validation is the determination of the correctness of the final program or software produced from a development project with respect to the user needs and requirements'; O'Keefe *et al.* (1987) note that 'validation means building the right system'. Validation is more concerned with the quality of the decisions made by the system.

Computer professionals concerned with development rarely deal with more than verification and validation, so called 'V&V', however this is really just the first stage in ensuring the quality of the system. An issue that determines the use of the system is that of credibility: the extent to which a system is believable or users can put credence in the system (Balci 1987). Other issues consider the 'fit' between the system and the user beyond the correctness of the decisions that the system makes. These issues (e.g., Buchanan and Shortliffe 1985) are summarized as assessment (O'Leary 1987), and include the nature of the discourse between the system and the user, the adequacy and efficiency of the hardware, the quality of the implementation, and the security of the system. Evaluation addresses the benefit of the system to the users, project sponsors and ultimately the organization, and is generally associated with measures of worth and value for money. In certain applications, issue of ethics and liability may also be important (Wyatt and Spiegelhalter 1990). Since the validity of an ES is often the key to its worth, evaluation is frequently misused for validation (e.g., Chandrasekaran 1983).

The relationship between these five aspects of the quality of a system form a hierarchy, shown in Figure 1. Essentially, each depends upon the level below it,

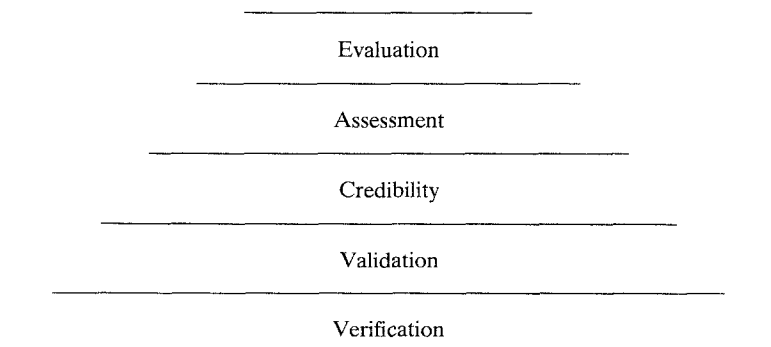

Fig. 1. A hierarchical view of system quality.

and in many cases problems at a lower level will result in problems at a higher level — e.g., a system that can not be shown to be valid may have little chance of being evaluated as worthwhile. Traditionally, different communities have concentrated on different parts of the hierarchy. Contrasted with V&V in computing, model building professionals, such as operations research scientists, have concentrated on validation and credibility. Often managers, project sponsors and end-users are only concerned with assessment and evaluation of the end-product.

2.2. *Verification, validation and credibility*

The types of errors that can occur in software can be regarded, at least partially, as a function of the technology used to implement the system. Thus with ES, the knowledge representation scheme and the method of handling uncertainty establishes much of the basis for verification. Validation generally is regarded as a more complex task that is not as dependent on the particular technology. Where a piece of software must perform in a pre-defined manner, such as a compiler developed for an ISO language standard, measuring validity is relatively straightforward. A compiler that incorrectly compiles any legal code is invalid. A different view of validation stems from the process of model building, particularly the construction of statistical, econometric and operations research models. A model is a representation of reality that will never be 'perfect'.

Landry *et al.* (1983) provide a conceptual framework for model validation. They define five types of validity:

1. *Conceptual*: 'the degree of relevance of the assumptions and theories underlying the conceptual model . . . for the intended users and use of the model'.
2. *Logical*: 'the capacity of the formal model to describe correctly and accurately the problem situation'.
3. *Experimental*: 'the quality and efficiency of the solution mechanism'.
4. *Operational*: 'the quality and applicability of the solutions and recommendations'.
5. *Data*: 'the sufficiency, accuracy, appropriateness, and availability of the data'.

Often model validation is an attempt to measure the generality of the model (for example, can it perform with different data sets or under differing assumptions). Typically, model validation is often equated with operational validity — a model is judged by the quality of its solutions — yet as Landry *et al.* point out, absolute validity based upon accuracy of solutions is a myth. The intended use of a model must be taken into account — a model may be valid for one application but not for another. Hence conceptual validation, and the criteria by which a model is judged, are both very important.

Since a model is never absolutely valid (or for that matter, absolutely invalid), its acceptance is very dependent upon the intended use and user. Hence model builders have recently started to focus more on credibility (for instance, Balci 1987; Gruhl 1982). It would be expected that the more valid a model the more credible it is, but there is no guarantee that a valid model will be perceived as credible.

2.3. *Assessment and evaluation*

Although the focus of this paper is on V&V it is important to understand the rest of the hierarchy. Verity, validity and credibility does not guarantee a useable, let alone useful, system. A system may be unusable, for example, because its response time is too slow or the interface is too complex. Such issues are certainly relevant when assessing an ES, but in general validation should be considered separately from assessment, not just as part of the assessment process (for instance, as done by Liebowitz 1986), since it is a prerequisite for adequate performance. A poorly validated system rapidly giving wrong advice through a beautifully designed interface may be dangerous: the superficial quality of the system may induce unwarranted credibility.

Traditionally, data processing systems have been evaluated purely on a financial basis, such as return on investment. A feasibility study specifies the costs and benefits of a system, and development is initiated or otherwise based upon these projections. Increasingly for many information systems, particularly decision support systems, the benefits are intangible. The value to an organization of improving the decision making process is often very difficult to quantify (O'Keefe 1989), and hence many systems are accepted for development, and subsequently evaluated, in an ad-hoc manner (Keen 1981; Hamilton and Chervany 1981).

Similarly, the benefits of many ESs are often intangible, and almost impossible to quantify. Hence formal evaluations that address the costs and worth of an ES are rare, although this may happen implicitly. Further, to this point in time one of the benefits of many developed systems has been increased familiarity with the technology for both the user group and the developers. Now that ESs are more pervasive, evaluation has become more stringent, and this in turn creates a need for more formal V&V (Batarekh *et al.* 1991).

3. CHARACTERISTICS OF EXPERT SYSTEMS

Different classes of software have particular characteristics that shape the way V&V is done. The dominant characteristic that makes ES V&V difficult is that an ES is simultaneously a piece of software *and* a model (Bellman 1990; Suen *et al.* 1990). Like any software, 'bugs' can cause unwanted and unpredicted consequences, but an ES is also a model of human knowledge and reasoning, and like all models, as mentioned above, will never be 'perfect'. Even if the software is completely verified and reliable, the embodied model may be in error. Divorcing the software and the model is difficult, if not impossible, although verification will normally concentrate on software aspects, whereas validation will concentrate on modeling aspects.

Viewing an ES as a model leads to a number of problems that affect V&V. One of the findings in the development of ESs is that understanding the domain is critical, as with most model building activities (Waterman 1986). In many other types of software it is assumed that the program specifications need only

be turned over to a programmer, who would then produce the code. However, generally, in order to produce an ES the developer must become a 'near-expert' in the particular domain (e.g., Lethan and Jacobson 1987). The domain itself defines what are the critical aspects of an ES developed in that domain: reasoning and knowledge in one domain may not be the same as that in other domains. Further, the expertise that is modeled in an ES is generally in short supply, is expensive, or is not readily available. This is in contrast to other types of software where there is substantial expertise available, such as some accounting systems, but is fairly common in model building activities.

In addition to the problems generated by the model characteristic of ESs, three other characteristics pose problems for V&V. First, ESs employ both numeric and symbolic information, rather than just numeric information — techniques typically used for the validation of numeric information may be infeasible, and this leads to a need for new validation methods. A purely quantitative model may produce a number that can be compared against an actual observation — the difference between the two is thus an estimate of accuracy. Contrast this with, for example, a page of textual advice representing a tax plan — comparison against an observed plan would require considerable expertise, and even then measuring the difference may be difficult. Kulikowski and Weiss (1982) discuss this problem in the context of the medical diagnosis system Casnet.

Second, ESs generally are developed using a 'middle-out' design, rather than a traditional top-down or bottom-up approach. A middle out approach starts with a prototype and gradually expands to meet the needs of the decision. Further, ESs have a tendency to evolve over time, as the decisions they model become better understood. Hence, as discussed later, knowing 'when' to actually validate an ES can be difficult.

Third, ESs are often used to model tasks for which computer programs have not been previously developed. Hence it is likely the task is not well-understood prior to the development effort (at least not by the developers) and thus there is no pre-established understanding of the problem. Thus defining performance criteria for the system, or developing a plan for V&V, can both be difficult to do early on.

4. VERIFICATION

Since construction of an ES is a programming task, traditional software engineering approaches to verification have a considerable role to play. The use of test data, the role of metrics and compliance to any written requirements are examples of verification issues well documented by others, particularly Adrion *et al.* (1982) and Boehm (1981). Here, for sake of avoiding repetition, only issues specific to ESs are covered. Rushby (1988) reviews software engineering approaches to V&V from the point of view of knowledge-based systems.

With ESs the primary verification efforts are likely to be aimed at the knowledge base, for at least two reasons. First, knowledge acquisition and

representation can be viewed as specific tasks, so that the knowledge itself needs to be verified irrespective of any coding or implementation. Second, in many cases an ES shell is used to develop the system, and here it is supposed that the inference engine and other facilities have already been verified by the shell developer (although, as discussed later, specifications of what the inference engine actual does are rare). Thus, only the knowledge and its representation needs verification.

Verification efforts typically try to exploit existing structure where feasible. Structure is either elicited from the domain (e.g., Davis 1984) or from the nature of the knowledge representation, and our discussion also will examine verification in the context of these topics. Since rule-based structures have to this point dominated ESs, it is no surprise that structural approaches to verifying ESs have concentrated on rules, particularly weights on rules (e.g., O'Leary 1990a) and the structure of the rules (e.g., Nguyen *et al.* 1987). Notions of completeness, consistency, correctness and redundancy permeate verification (Adrion *et al.* 1982), and hence both domain dependent and independent approaches have tended to categorize the methods used under these headings.

4.1. *Domain dependent verification*

Domain dependent verification employs meta-knowledge from the domain to examine the verity of the knowledge. The earliest and best known example of domain dependent verification is the work of Davis on TEIRESIAS, which verifies the addition of new knowledge to MYCIN (Davies and Lenat 1982).

In order to illustrate domain dependent verification of knowledge consider the example of a living room. Meta-knowledge could be used to ascertain that knowledge about a specific living room would be *incomplete* if the living room did not contain a couch. Similarly, there may be *redundant* knowledge if it is suggested that there are two couches in the living room. In addition, it probably would be *incorrect* if the knowledge placed a bathtub in the living room. Finally, it would be *inconsistent* if we labeled two physically different items in the living room with the same name, e.g., calling both a couch and a large chair the 'couch'.

Domain dependent approaches frequently are embedded in the knowledge acquisition process and tools that support knowledge acquisition (e.g., Boose and Bradshaw 1987; Gaines 1987). Such systems often build the user into the verification process as a source of meta-knowledge. In addition, as the systems accumulate knowledge about the domain, they can use that knowledge to verify new knowledge gained as part of the knowledge acquisition process.

Domain dependence has many limitations. First, with a domain dependent approach, verification must also be made of the meta-knowledge. If humans furnish a major portion of the meta-knowledge, then problems that are typical of human users, such as inconsistencies, can permeate the system. Second, the meta-knowledge may not be stable, and it may not be practical to frequently update it. Third, in general the development of a meta-knowledge approach

may be quite costly due to the costs of acquiring and maintaining the meta-knowledge, which may be different for differing systems. For these reasons, most approaches to verification are domain independent: they analyze structure independent of knowledge about the domain.

4.2. *Domain independent verification*

Domain independent approaches are typically based upon the concept of an *anomaly*, where an anomaly is an abuse or unusual use of the knowledge representation scheme. An anomaly can be considered a *potential error* — it may be an actual error that needs correcting, or may alternatively be intended. Some anomalies simply cause efficiency or maintenance problems. Verification based upon anomaly detection is a *heuristic* approach (Miller 1990), rather than deterministic, for two reasons. First, detected anomalies may not be errors, and errors may exist that are not related to known anomalies. Second, some of the methods for detecting anomalies are themselves heuristic, and thus do not guarantee detection of all identifiable anomalies.

4.2.1. *Rule verification: structure*

Considerable research has been done on identifying rule-base anomalies, including Suwa *et al.* (1982), Nguyen *et al.* (1985, 1987), Nazareth (1989) and Preece *et al.* (1992), with the result that rule anomalies are now quite well understood. Thus here we briefly review what we know about rule anomalies, but avoid the details.

Consistency generally means to call the same attribute by the same name or the same conclusion by the same name. Since humans have a tendency to make errors in typing or inconsistently call the same item by different names, procedures need to be adapted to ensure that names are consistent. Within a rule-base, consistency is often best implemented by requiring that the developer establish sets of lists of attributes and conclusions from which rules can be constructed.

Completeness checks cannot determine that there is complete knowledge in the system, rather, using some of the structure associated with rules it is possible to determine some situations where there likely is incompleteness. That structure includes the lists of attributes and conclusions established to construct the rules, the fact that each rule must have both attributes and conclusions, and that each rule must either lead to another rule or a terminal conclusion. Specific identifiable completeness problems include unreferenced attributes and conclusions, illegal attributes or conclusions, unreachable premises and deadend conclusions.

Correctness is the identification of the structure of the rule-based, and ensuring that none of the structure is violated. Examples include conflicting rules (where two or more rules have the same *if* attributes, but come to contradictory conclusions), subsumed rules (where two or more rules have the same conclusions, 'but one contains additional constraints on the situations in which it will succeed', Nguyen *et al.* 1987), and circular rules (where there

exists a chain of reasoning that starts with some condition and then returns to that same condition).

Whereas a circular reasoning chain may be detected in execution, and some shells allow conflicting rules as a way of providing multiple conclusions or accumulating certainty factors for different conclusions, subsumed rules are more insidious, and are worth dwelling on (particularly as we will return to them when we consider hybrid systems). Consider the two rules:

(1.1) $P(z) \,\&\, Q(z) \,\&\, R(z) \rightarrow S(z)$

(1.2) $P(x) \,\&\, Q(x) \rightarrow S(x).$

Rule 1.1 is subsumed in rule 1.2 (using the definition of Nguyen *et al.* 1987), yet rule-base developers often do this, for at least two reasons. First, particularly when forward chaining is used, a more specific pattern match should be attempted first, followed by less specific matches. Rule 1.2 can be interpreted as 'if we don't know $R(z)$, do $S(x)$ anyway'. Second, when uncertainty weights are used, we may conclude the more specific rule 1.1 with greater certainty.

Redundancy is where two or more rules 'succeed in the same situation and have the same conclusions' (Nguyen *et al.* 1987), or the reasoning chain contains a redundant rule, such as

(2.1) $P(x) \rightarrow Q(x)$

(2.2) $Q(x) \rightarrow R(x)$

(2.3) $P(x) \rightarrow R(x)$

where $Q(x)$ does not appear anywhere else in the rule-base. As noted in Suwa *et al.* (1982) redundancy has an impact on efficiency, but does not necessarily cause logical problems.

4.2.2. *Rule verification: weights*

In systems which attempt to measure uncertainty or strength of association, using certainty factors, Bayesian probabilities or any other method, it is also important to verify that the weights are consistent, complete, correct and not redundant. This can be done by ensuring that each rule that is supposed to have a weight does have one and that the weights are developed in concert with the theory on which they are based.

Finding anomalies in the weights in an ES is a process that has received limited attention, probably due to the limited number of implemented ES that make extensive use of uncertainty measures. O'Leary (1990a) developed some verification results using an analytic approach for a Bayesian-based uncertainty scheme. Applying those results to a number of published papers on ESs with uncertainty factors found that almost all systems had implemented the approach in error or had developed weights that were 'unusual'. Lehner (1989) developed a nonparametric procedure to test the effect of individual nodes in an inference network on the hypothesis considered by the network. Yager and Larsen (1991) developed a novel approach which runs the inference 'backwards' so as to

determine the allowable values for required input; where this conflicts with the developers understanding of the system, errors may exist in the representation of the weights.

4.2.2.1. *The Bayesian approach.* Since the Bayesian approach, originally conceived for PROSPECTOR (Duda *et al.* 1979), and now implemented in a number of shells (for instance AL/X and Savoir) is common and relatively long standing, more is known about weight anomalies under this scheme than any other. In the Bayesian approach knowledge is specified as a set of rules and weights of the form

if E then H (to degree S, N)

where S and N are numeric values that represent the strength of association between E and H. S is a sufficiency factor, since a large value of S means that a high probability for E is sufficient to produce a high probability for H, and N is a necessity factor, since a small value of N indicates that a high probability of E is necessary to produce a high probability of H. S and N can be specified directly, or they can be developed by establishing the probabilities in the likelihood ratios:

$$S = \Pr(E \mid H)/\Pr(E \mid H')$$
$$N = \Pr(E' \mid H)/\Pr(E' \mid H'),$$

where E' is 'not E; and H' is 'not H'. In some implementations, logarithms of these values are used, referred to as NW and PW, and are scaled between -100 to $+100$.

Given these definitions of PW and NW, it is easy to show that there are deterministic relationships between the weights. These include:

$PW > 0$ if and only if $NW < 0$,

$PW = 0$ if and only if $NW = 0$,

and

$PW < 0$ if and only if $NW > 0$.

Anomalies exist if these are violated. A number of systems apparently have weights that violate these rules (O'Leary 1990a).

In systems where the weights are solicited directly, rather than through the use of probabilities, relationships between pairs of PW's and NW's could be developed, whether consciously or unconsciously. Those relationships could take any of a number of functional forms, including linear, quadratic, etc. Empirically, it appears that developers have frequently employed a relationship of $PW = -NW$. It seems that there is a desire to capture symmetry of uncertainty of strength of association through symmetry in the weights. Thus, in this case developers have established a linear functional relationship between the pairs of weights. Unfortunately, symmetry with the weights expresses an unusual nonsymmetric requirement on the underlying probabilities. Doing some

basic analysis we can see that the requirement that $PW = -NW$ implies that

$$\Pr(E \mid H) - (\Pr(E \mid H))^2 = \Pr(E \mid H') - (\Pr(E \mid H'))^2,$$

a less than intuitive result. This requires that the probabilities be drawn from two intersecting lines. Thus, where the weights are likelihood ratios (e.g., *PW*'s and *NW*'s), such asymmetry can lead to 'unusual' behavior for the underlying probability.

4.2.3. *Statistical investigations*

Statistical methods can be a useful static or dynamic verification test. Landauer (1990) and O'Leary (1988a) suggest that various aspects of rules, such as attributes and conclusions, be analyzed statistically as part of the verification process. This can be done statically or dynamically. A simple tally of attribute occurrences can give insight, for example, suppose that the attribute *new_cash_flow* appears 32 times in a knowledge base, and *new_cash_flows* appears only once. This suggests that either *new_cash_flows* is a misspelling or is a different rare attribute. Similarly, suppose that *cash_flow* appears in a rule with *present_value* 10 times, but on its own only once. Either these are linked attributes and one rule is in error, or *cash_flow* can occur separately.

O'Leary (1988a) suggests statistical analysis of the frequency that rules are fired or paths are traversed. For example, a priori, it may be expected that a particular rule or sequence of rules should fire frequently. If analysis of actual or simulated use of the system provides data that indicates that this is not the case, then it would be appropriate to examine those rules in more detail.

In some cases we might be able to make certain assumptions about the weights used to represent uncertainty in ESs. O'Leary and Kandelin (1988) developed and illustrated some of the verification issues that can be identified using this approach. For each of these approaches, statistical methods are developed or discussed. There are theoretic reasons for the likelihood ratios to be normally distributed, and thus one verification test is to determine if the weights are normally distributed. On interlinking sets of weights such as those in PROSPECTOR and many subsequent systems, if one weight changes then so should the other, and thus we would expect changes in a distribution of *PW*'s would lead to changes in the distribution of the *NW*'s. Further, if our understanding changes from one version of the system to another, then we should expect to find changes in the distribution of the weights from one version to another. In O'Leary and Kandelin (1988) the correlation of the *PW*'s and *NW*'s was explored to find that the system under consideration had a very unusual correlation, suggesting that the weights were constructed in an inappropriate manner.

4.2.4. *Hybrid systems*

Increasingly, implemented ES employ hybrid tools that use some variation of object-oriented methods to store attributes and procedural attachments and provide inheritance. This considerably alters the anomalies that can occur with rules, and introduces other anomalies that need to be considered.

Tools such as KEE and NEXPERT use a hierarchical frame structure, such that diagrams of frame-based systems take on a tree-like or an acyclic network structure. The 'top' frame can be referred to as the root frame; the 'bottom' frames can be referred to as terminal frames. We can consider the meaning of consistency, completeness, correctness and redundancy in terms of frame contents and structure.

Consistency of frames refers, in part, to the names and labels given to the slots and contents. One approach is to require the development of lists of names for frames, slots and contents. System construction would then require choice from those lists.

Incompleteness can occur if there is a missing frame, or a missing slot or a missing link between frames: (1) if a frame is a root frame and there is no path out then there is a missing link, (2) if the frame is a terminal frame and there are no links into the frame then there is a missing link, and (3) if the frame is an intermediate frame then it should have at least one link in and at least one link out. If the frame names, slots and contents are established as lists, as noted above, then those lists can be used to establish tests of completeness, as with rules. For example, each name (frame or slot or content) should come from a list. Further, if it is on a list, then it should be used in some frame, slot or content, otherwise it is incomplete.

Correctness in a frame system will continue to be mainly dependent upon the structure of the rules. Inheritance has a major effect on correctness, and thus anomalies additional to those discussed earlier can be present. This is particularly true with subsumption, where what has been called *hierarchical subsumption* (Lee and O'Keefe forthcoming) can now occur. Consider the example:

$$(3.1) \quad P(x) \,\&\, Q(x) \rightarrow S(x)$$

$$(3.2) \quad P(x) \,\&\, Q'(x) \rightarrow S(x)$$

where $Q'(x)$ is an instance of $Q(x)$ (for example, Q' may be 'sports car' and Q is 'car'). Rule 3.2 is subsumed in rule 3.1 since is it less specific, and hence 3.1 can always fire when 3.2 can. This problem can be checked for, and at least one tool (CLASP, Yen *et al.* 1991) prohibits this via careful identification of subsumption relationships.

Redundancy can occur in at least four different ways in frame systems: redundant frames, redundant slots within a frame, redundant contents within a frame and redundant connections with other frames. Redundant frames could be determined by a straight comparison of the contents of frames with each other. Redundant slots can be determined by comparing slots within a frame to each other, and redundant contents can be determined by investigating contents for such redundancies. Redundant connections can be determined by examining the connections to other frames, from a given frame (O'Leary 1990b).

In the same sense that weights, attributes or rule firings can be investigated statistically, so can characteristics of frames. For example, slot contents can be analyzed for frequency of appearance as single items or paired with other slot contents. The frequency that particular frames are employed could be dynami-

cally accumulated; if a frame is used much less than expected then that could indicate a problem in the tree structure of the knowledge base.

4.3. *Errors in the inference mechanism*

Particularly when using a shell, the developer normally assumes that the inference mechanism is 'correct', i.e., behaves as expected. This assumption may not always be warranted, particularly for newly introduced versions of a shell. For critical applications, faith in the inference mechanism should be established via testing. Where a developer is building an inference mechanism, or enhancing an existing one, then this will have to be verified separately from the knowledge base. This should normally occur before verification of the knowledge base.

Conflict resolution procedures, and tools such as OPS5 that provide a number of different conflict resolutions strategies, can be a source of problems. If the developer does not fully understand the procedure, then although the static knowledge-base is verifiable, the system may perform in an unexpected manner. Inheritance also presents difficulty, since with tools that provide for multiple inheritance the exact course of inference can be difficult to ascertain. Standards are necessary here, so that a named method can be guaranteed to perform in a certain way (Harrison and Ratcliffe 1991). Further, many identified anomalies, and automated tools that search for those anomalies, discussed below, make stringent assumptions about inference.

4.4. *Automation*

More than other software, ESs offer possibilities for automating the verification process. This is because the separation of inference and knowledge means that the knowledge base should exhibit completeness and consistency irrespective of the inference mechanism. Current automated tools analyze a knowledge base by either filtering knowledge through meta-knowledge (domain dependent), or by converting the knowledge base to an intermediate representation, such as a table or graph, and searching for anomalies (domain independent).

The best example of the meta-knowledge approach is the Expert systems Validation Associate (EVA) (Chang *et al.* 1990), 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 a structure-checking algorithm and previously developed meta-knowledge. As an example, EVA provides a meta-predicate called *incompatible*. The meta-knowledge

$$incompatible(\text{party}(X), \text{at}(\text{bob}, X), \text{at}(\text{wife\_of}(\text{bob}), X))$$

states that "bob and his wife can not be at the same party". The major problem with EVA is how to categorize subjects and reduce the meta-knoweldge base: without meaningful context, the meta-knowledge base may become larger than the actual knowledge base.

Early domain independent approaches employed condition/action tables,

which separate rules' condition and action parameters. Algorithms then examine the existence of relationships among the rows and columns. Examples include the Expert System Checker (ESC) (Cragun and Steudel 1987) and the Rule Checking Program (RCP) (Suwa *et al.* 1982). On the other hand, Nguyen *et al.* (1987) uses a dependency chart to detect a circular rule chain. The CSRV (Cross Reference, Style & Verification) toolset for the CLIPS shell developed at NASA (Castore 1987) provides similar support via cross referencing of parameters and relations in the rule base. VALIDATOR (Jafar and Bahill 1990) provides some of the statistical analysis checks discussed earlier.

Preece's COVER system (Preece 1989, 1990; Preece *et al.* 1992) improves considerably on table-based methods by constructing a graph representation of the rule base directly from the rules. This has the advantage of detecting anomalies across numerous rules, rather than between pairs of rules, as is common with the table-based approaches. KB-Reducer (Ginsberg 1988; Ginsberg *et al.* 1988) is also a major advance over table construction methods: rules are transformed into a logical form, and so called *labels* are generated that express the conditions under which each hypothesis is true. As each label is generated a truth maintenance approach is used to check for redundancy, contradictions and inconsistency (Zlatareva 1992, gives an extensive review of the actual methods). Theoretically, KB-Reducer can detect all potential contradictors in an object-attribute-value rule base for an inference mechanism that is (1) monotonic, (2) does not use any conflict resolution strategy, and (3) is data-driven in the sense that any required data is available in working memory.

Present automated tools are all limited to 2-valued logic production systems that employ certain reasoning, and, as discussed above and identified for KB-Reducer, make stringent assumptions about inference. With the development of more hybrid systems, advances are necesary to take account of object-oriented representations. One recent development that attempts to address the order in which rules are fired (and thus makes less assumptions about inference and conflict resolution) is approaches based upon Petri nets (Agarwal and Tanniru 1992; Liu and Dillon 1991). Groups of rules can be represented as a Petri net, which can then be tested for completeness using existing methods. It is also suggested that this approach can handle temporal relationships between rules. However, conversion of a rule base into a Petri net is a non-trivial task.

Another problem with domain independent approaches is that they face a combinatorial explosion as the number of attributes increase: Ginsberg (1988) reports that KB-Reducer took 40 cpu seconds on an Explorer II to analyze a base of 30 rules, but 10 cpu *hours* to analyze a base of 370 rules. Attempts to overcome this include partitioning the rule base, as is done in ESC, and the use of heuristics, as is done in COVER.

4.4.1. *The value of automated tools*

A lot of research effort has been put into the design and development of automated tools, and thus it is worth dwelling on their value and impact. Whether or not they become commonly used is debatable, but one school of thought suggests that knowledge engineers should aim to design correct con-

sistent knowledge bases rather than rely on automated tools to weed out the errors in poorly designed bases.

To this end, a number of shells provide built in verification checks, particularly concerning attributes and attribute values. For example, some shell languages require that attributes be declared and the rule compiler will signal undeclared attributes. This greatly reduces the number of consistency and completeness anomalies that can occur, but can have the effect of making the shell language appear more like a third generation language than a symbolic tool. Another common feature is the ability to specify mutually exclusive attribute values, for example to state that an attribute PREGNANT can not be true if the object has an attribute MALE which is also true. This, and other facilities, can be viewed as simple steps towards the incorporation of meta-knowledge for verification. Facilities in this area are likely to increase in future generations.

Another allied development is means to partition a rule base, mainly based upon between rule distance measures, such as Jacob and Froscher's (1990) method. These can result in errors and side-effects being localized, and hence far easier to control. Alternatively, partitioning also makes use of an automated tool easier, as discussed above overcoming combinatorial problems.

5. THE STRUCTURE OF VALIDATION

As discussed earlier, validation is inherently more complex than verification, dependent upon the criteria on which the system is to be judged and its intended use. Hence the validation process should be properly structured: the means whereby the system will be declared as valid or otherwise should be established at the outset of the process. This is an important part of any specification (Batarekh *et al.* 1991).

This paper uses a framework, based on the theory of research methods, to investigate the structuring of validation efforts. The framework covers *establishing criteria* for validation, *criterion vs. construct* validity, *maintaining objectivity* and *reliability.*

5.1. *Establishing criteria*

5.1.1. *Level of expertise*

The simplest approach to establishing the criterion or criteria for validating a system is to define the output level of expertise that the system should perform at. It may be required that a system performs at the level of an expert, better than an expert, or at the level of a good trainee. For example, the performance of both ONCOCIN and MYCIN were shown to be reasonably close to experts at the Stanford medical center where the systems were developed (Hickam *et al.* 1985; Yu *et al.* 1979a, b); it is sometimes stated that the mass spectrometer

analysis program DENDRAL, also developed at Stanford, performed at the level of an organic chemistry Ph.D. (Buchanan *et al.* 1969).

In some instances it will be required that the system perform better than an expert. This may be achievable, for example, by the system being more consistent, not tiring, being able to deal with more data, being able to solve problems more quickly, etc. Hence criteria concerning consistency, speed, etc. may dominate criteria concerning, for instance, quality of solutions. A good example of this is the R1/XCON system (Bachant and McDermott 1983), where previous human performance was very mixed and introduction of the system established some much needed consistency. Finally, in other situations, in order for the system to be usable it must function at least at some minimal level of performance. One view of that minimal level is that of a trainee or some other nonexpert level.

Two suggested methods for defining the level of expertise of a system require mention if only so that they can be subsequently disregarded. The first method is to get the expert on whose knowledge the system is based to 'sign-off' on the system, stating that it performs at their level. This is obviously convenient from the viewpoint of legal liability: whether it validates a system is very debatable. The second is to get the system to sit an exam, since many professions, such as accounting and medicine, have entrance exams that test basic competence. Such exams, however, generally test breath, whereas most systems developed so far are intentionally narrow in their specialty domain. In addition, typically such exams are minimal levels of performance rather than measures of expertise within those professions.

5.1.2. *Performance range*

A convenient approach to measuring the level of expertise of a system is to judge its success in solving problems. The system will not necessarily perform at the expert level, although it might, but perform within some range. The performance acceptable to users and sponsors is called the *acceptable performance range*; if a minimum level of competence is defined, this is sometimes called the *acceptable level of performance* or ALP.

Many developed ES have explicitly used level of performance as an evaluation criterion. Typically, a number of case studies are presented to the system, and the number of 'correct' answers, compared to those of an expert, are tallied. The system is then determined to be, for example, '90% correct' or '95% perfect'. Such figures may be meaningless: they are simply a function of the cases presented to the system. What is more, in many instances a large number of cases dealt with by an expert are 'standard', and the leverage of the expert is determined by the difficult and obscure problems that are faced.

As an example, consider the case of a system designed to evaluate firms for whether or not they will go bankrupt. Typically the success of such a system should be in its ability to find those relatively few firms that will go bankrupt. In any given year in the US, roughly 95 to 98% do not go bankrupt, and hence it is conceivable that a system may correctly categorize 95% of the total without being able to determine *any* of the firms that actually went bankrupt.

5.1.3. *Builder's/user's risk*

A more formal way to specify criteria for a performance range developed by Balci and Sargent (1981) is based upon the standard statistical concept of Type I and Type II errors, as shown in Figure 2. A Type I error results if a system is rejected as invalid when it is in fact valid; a Type II error results when an invalid system is accepted as valid. The probability of the first is *builder's risk*, since the effect is to prolong development of an acceptable system, and perhaps even abandon development. The probability of the second is *user's risk*, since the effect may be dramatic for the user who accepts incorrect results, such as loss of life or large sums of money.

		State of the expert system	
		System is valid	System is invalid
Action	Accept as valid	Correct decision	**User's risk** (Type II error)
	Declare invalid	**Builder's risk** (Type I error)	Correct decision

Fig. 2. Type I and Type II errors in validation.

For many ESs it will be impossible to quantify either risk. However, for all systems it should be possible to consider the *relative* importance of each risk. For example, with R1/XCON a high user's risk was acceptable, due to the relatively poor performance of the 'human predecessors', but with medical systems user's risk must be shown to be virtually zero. This is, of course, very difficult to do. For systems with a limited set of outcomes, such as many straightforward classification systems, relative risks can be expressed at the level of actual outcomes. Suppose that an ES produces a classification as A, B or C. It may be that certain variations in performance are acceptable, but others are not. For example, an incorrect classification of A as B may be acceptable, but A as C may not. Hence in validation it must be shown that prob($A \mid C$) is zero, but prob($A \mid B$) can be greater than zero.

5.2. *Criterion vs. construct validity*

All of the various approaches to establishing criteria discussed above are in fact variations of what social science model builders call criterion validity. 'Criterion validity is studied by comparing test or scale scores with one or more external variables or criteria, known or believed to measure the attribute under study' (Kerlinger 1973). The attribute is expertise; the criteria is some variation of performance range. An alternative type of validity is construct validity: validation against the theory on which the system is based. As noted by Kerlinger

(1973) 'the significant point about construct validity that sets it apart from other types of validity is its preoccupation with theory and theoretical constructs'.

With the majority of ESs the developer elicits the knowledge base in a purely empirical manner, where knowledge is treated as something to be iteratively discovered without reference to any underlying theory to guide the investigation. Hence criterion validity has dominated ES validation, and will likely continue to do so. Construct validity may have an increasing role to play, however. Recent developments in qualitative and causal reasoning have resulted in systems based upon 'first principles' derived from an understanding of the causality in the domain being examined (for example, Bratko *et al.* 1989; Davis 1984).

There are at least three potential validation comparisons associated with construct validity: (1) the comparison between the system and the first principles, (2) the comparison of the first principles to an expert, and (3) the comparison of the system to the human expert. Since (1) and (2) can be difficult to do, (3) is normally preferred; one elegant way to do this, as used by Bratko *et al.* (1989) and Pearce (1988), is to induce a shallow model from the deep model using induction techniques, and then compare both the structure and performance of the shallow model to the expert.

5.3. *Maintaining objectivity*

Verification investigates aspects that are not open to subjective appraisal, so objectivity is generally not an issue (although interpreting when an anomaly is an error can be subjective). On the other hand, objectivity in validation is critical, since the measurement of validity against established criteria may be open to interpretation. Ideally, validation tests are built into the software, similar to automated verification tests, and particular procedures are implemented without intervention. For all but the simplest of validation criterion, this is impossible to do; hence it is necessary to check the objectivity of the human validator.

Typically the knowledge engineer is continually verifying and validating the system, based on skills that are brought to the system and gained during system development. However, those efforts may be less than objective due to a number of factors. If the developer is short on time or budget, the validation effort may be cut, since it may be seen as an overhead function. If the developer has a vested interest in the system, then letting the programmer instigate the only V&V procedures is somewhat analogous to letting the 'fox guard the henhouse': there is a potential for substantial violations of objectivity. In order to mitigate such violations, typically research methods emphasize the importance of the *independence* of the validator.

With conventional software, software engineers often use an acceptance test by the sponsor or end user as the final step in the validation process. If validation criteria are well established, as discussed earlier, then a validation by the sponsor or end user provides evidence that at least the system meets those criteria. Unfortunately, if a system changes substantially, then the notion of an

acceptance test may be difficult to implement. Further, as noted in O'Leary (1987), in some cases the end user or sponsor may have insufficient expertise to validate the system. Hence the developers *have* to establish validity, and then build system credibility by professionally reporting the details of the validation process.

To overcome these problems, an attractive approach is to get 'third-party' experts, i.e. experts not involved with the development effort or not even part of the sponsoring organization, to validate the system. Some researchers, for example Buchanan and Shortliffe (1985), have reported that in some cases external validators may be biased in their validation of a system if they know that the problem solution was generated by a computer. They may expect less from the system, or more likely have unreasonable expectations of performance (this is called 'the super-human fallacy' by Chandrasekaran 1983). Other biases may also influence evaluators. If a human investigator finds that a problem solution is from a rival expert, or simply one from another 'school of thought', then that could have an impact on the quality they attribute to the solution. Clearly, this means that blinding techniques must often be used to mitigate violations of objectivity.

5.4. *Reliability*

In addition to performing within an acceptable range, a system must be a *reliable* representation of the expert's knowledge. The expert's knowledge is captured by the knowledge engineer, who then casts his representation of that knowledge into a computer program. There are two possible interpretations of reliability here. In the first, total reliability occurs when the knowledge 'reported' by the expert and the actual knowledge of the expert are the same. In the second, total reliability occurs when the knowledge 'reported' by the expert and the knowledge in the computer program are the same. One perspective on this notion of reliability is that the uncertainty of capturing knowledge cascades from one representation to another.

Based on this concept, O'Leary (1988b) developed an analytic Bayesian model to investigate the impact of cascaded reliability on weights in uncertainty measures. In this case the concern is with the relationship between an actual event, a report of the event and the corresponding weights in an ES that relate to the evidence. A conclusion from this work is that even a small loss in reliability can result in the weights in the system being considerably different from the evidence. Thus mitigation of the cascade effect is crucial if the ES is to be a valid representation.

6. VALIDATION METHODS

Once criteria have been established, the goals of validation determined and the various problems addressed, appropriate methods have to be chosen. The criteria, existence of a theory and pragmatic issues such as the availability of

experts, case studies, time and money will determine the appropriate methods. ESs can be validated by either examining the individual components, such as the rules, weights or frames, or by examining the operation of the system as a whole. In the later case it can be treated as a black box simply to determine if it is making the right decisions, or it can be opened up to determine if the line of reasoning is correct, i.e., it is making the right decisions for the right reasons (e.g., O'Leary 1988a).

6.1. *Component validation*

6.1.1. *Rule validation*

A common method of component validation is to directly examine the knowledge base so as to assess the accuracy, representativeness and validity of individual rules. The process of direct examination is facilitated by the manner in which some ES shells allow the user to access the knowledge.

Where the knowledge base is too large for a complete direct examination, approaches can be used to choose which rules are more important to examine (O'Leary 1988a). For example, those rules that have the most costly consequences or largest profit generally should be investigated. When using an uncertainty measure, those rules with either the larger or smaller weights should be examined, since they have the greatest impact on the solution generated by the system. Another approach is to determine which rules fire the most (or the least) in simulated paths through a rule-base, and examine these for their quality, since they either are frequently or rarely in the solutions generated.

The value of any sort of rule validation is debatable. If elicitation was based around generating suitable rules, then their correct representation in the rule-base should have been asserted as part of the verification process. Performance problems are more likely to arise from missing rules or unforeseen interactions between rules.

6.1.2. *Heuristics*

An ES, particularly a rule-based system, can be viewed as a collection of heuristics or a single large heuristic. A single rule may be a sensible heuristic in its own right, or, more likely, a combination of rules can be considered as a complete separate heuristic producing a solution given input. A modular system may be composed of a number of such heuristics.

Previous work in validating mathematical heuristics, reviewed by Eglese (1986), provides a method for validating knowledge-based heuristics. Many mathematical heuristics are used where existing optimization methods are too inefficient for solving problems on a regular basis or in real time. In these cases the results from a heuristic can be compared against the optimum solution, and deviation from the optimum can provide a measure of the 'goodness' of the heuristic. Similarly, where an ES works on a problem where complete enumeration of the state space or an optimization method can provide an optimum result, this comparison can be made. Some constraint reasoning systems, such

as those developed to produce production plans and schedules (for example, Ow and Smith 1987), fall into this category. A pit-fall here is the 'scale up' assumption — typically a comparison will be made based upon small versions of the problem, and it is assumed that validity scales up to larger versions. This is reasonable, but by no means certain.

Heuristic validation also uses a concept called *worst case analysis*, which is the maximum deviation from the optimum that can ever occur. For some mathematic heuristics the worse case can be proved; for knowledge-based heuristics this is unlikely to be possible, but experimentation with numerous problems, or a detailed analysis of the search mechanisms, and comparison against known results may provide an estimate of the worst case.

A complicating factor in knowledge-based heuristics is uncertainty measures where an uncertain estimate input by the user or produced by another heuristic affects the outcome. If the outcome is itself a measure of uncertainty, then this will vary over some range depending upon the estimates input. For example, as the input varies from -1 to $+1$, the output will itself vary over all or some part of the range -1 to $+1$. Langlotz *et al.* (1986) show how this situation can be regarded as a distribution sampling simulation, where samples of the input produce a distribution for the output. This can then be used to aid investigation of the validity of the heuristic.

6.1.3. *Meta-models*

A meta-model expresses the relationships between the elements of a model: it is a model of a model. Where a knowledge base becomes large it can be useful to have a model of the constructs and concepts present: this can then be used to determine conceptual validity. Rushby (1988), following Pearce (1988), suggests that a rule base should be generated from a causal model, or at least that a causal model should be maintained in parallel with the rule base. Expressed in diagrammatic form, a causal model can then be used by developers and experts to check for completeness of the knowledge base. Unlike the previously discussed automated verification tools, which relate rules to each other, a causal diagram relates the concepts that are expressed as rules; as an example, Figure 3 shows two rules concerning automobile fault diagnosis and an associated causal link.

Meta-models are very useful for understanding large rule-bases, but if

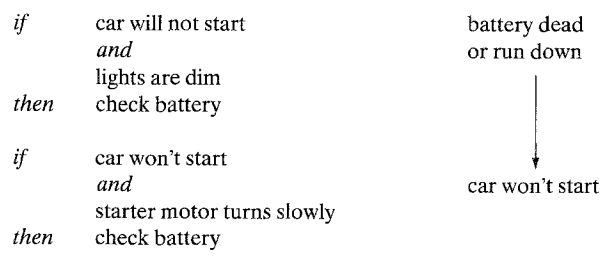

Fig. 3. Using a causal diagram to express concepts in rules.

validation focuses on the intermediate causal representation, then errors in its implementation at the rule level may be missed.

6.2. *System validation*

6.2.1. *Test cases*

Based on a survey of ES developers (O'Leary 1991), using test cases seems to be the present dominant method for the systematic validation of ESs. Cases previously solved by an expert are run through the system, or new cases are presented to both expert and system, and the solutions are compared.

There are at least four guidelines that should be followed when selecting test cases. First, the problems to be encountered by the system should be reflected in the cases (Chandrasekaran 1983). Using terminology from software verification, this implies that there should be a *prescribed input domain*: the boundaries of the input that the system will receive should be specified. Second, a sufficient number of test cases is necessary to elicit the range of parameters necessary to test the system and to be able to establish some statistical measures of significance. However, as noted by O'Keefe *et al.* (1987) 'the issue is the coverage of the test data — that is, how well they reflect the input domain', not the *number* of cases that are used. A sufficient variation in the test problems is necessary to test the range of parameters in the system.

Third, the nature of the problems investigated by the system should help establish the characteristics of the cases. Returning to the example of a system designed to investigate bankruptcy, in any one year roughly 2 to 5% of US firms go bankrupt. Thus, if a system was given test data in proportions to the occurrence of bankruptcy in the actual population (e.g., 96% not bankrupt and 4% bankrupt) the non bankrupt firms would flood the system to result in a high success rate.

Fourth, in some domains expert decisions may precipitate actual outcome. O'Keefe *et al.* (1987) gives an example: 'suppose that a bank uses a performance prediction when deciding whether or not to support company X financially. If an expert had decided a year ago that the financial position of company X would be poor in a year's time and thus implemented withdrawal of financial support, the present poor financial position of company X might be due in part to that previous expert position'. Choice of test cases is thus difficult, if not impossible, in such domains. King and Phythian (1992) present a case of a system that supports the decision to determine whether or not to tender a bid for a contract, where being awarded the contract on past cases can not be used as an indicator of success since quite obviously the company would not obtain any contract unless they bid for it.

A problem with using test cases is an assumption that the expert against which the system is being compared is *always correct*, i.e. if the system differs from the expert then it is 'wrong'. This, quite obviously, is not always the case. One of the authors was involved in the development of a personnel selection system, which underwent extensive validation, where it was realized that in a

number of the test cases the previous 'expert' had missed or misinterpreted something, and hence made an incorrect decision. When the system made a correct decision in these cases, the credibility of the system was greatly enhanced. Another potential problem is that in many instances test cases will not be available. Synthetic cases can be produced, but this is dangerous, and demands considerable objectivity on behalf of the validators. There is a temptation to make the cases reflect the known strengths of the system.

Despite the above problems, case testing has tremendous appeal. Small or limited cases can be useful early on in development, and the case load can be broadened as the ES matures. Extensive alterations to the ES can be tested for side-effects by running cases through it that were previously solved. In this way, case testing nicely fits the evolutionary development method common to so many ES.

6.2.2. *Turing tests*

In the classic Turing test (Turing 1950), a third-party has access to the output from both machine and human, and has to determine which is which. As a validation tool, a Turing test refers to a third-party expert comparing the results from an ES with those from a human expert. To overcome the objectivity problems discussed earlier, the process should be blinded so that it is not clear which is the computer's and which is the human's.

Test cases are necessary for Turing tests, and the discussion above equally applies. Now, however, *there is no assumption that the human expert is correct*: the third-party expert can compare, rank or criticize as deemed appropriate. For many ESs, Turing tests are the most appropriate validation method. They are particularly useful when (1) it is difficult for the developer to assess output on a case study as correct, or otherwise, or make judgments about how it differs from a human expert (this is often the case when output is holistic, and difficult to quantify), or (2) the system must be validated against multiple experts and there is variation between the performance of the experts.

MYCIN was validated using a Turing test methodology (Yu *et al.* 1979b; Buchanan and Shortliffe 1985). Ten cases were developed and analyzed by 10 'experts', including the system. These 100 case results were then evaluated by 8 evaluators, using one of three alternatives on a rating system, to establish a level of performance for each expert. The rating system alternatives were 'equivalent' (i.e., the evaluator would have done the same thing), 'acceptable alternative', or 'not acceptable'. Hickam *et al.* (1985) discuss a similar Turing test validation of the chemotherapy advisor ONCOCIN.

Hansen and Messier (1986) discuss a test of the auditing ES EDP-XPERT, and a second more extensive test is reported in Messier and Hansen (1992). EDP-XPERT provides a certainty measure for the electronic data processing controls in computerized accounting systems, producing three measures representing confidence in the supervisory, database management and application controls. In each test, expert auditors and the system produced a measure, and these were compared by the developers. Interestingly, the experts then had the opportunity to produce a second measure given the output from the system. In

the first test, where the 'experts' were low level computer audit specialists (in the second test they were senior experienced specialists), a significant number changed their answer. This may be a useful approach to validating systems that will be used in a supporting role.

Despite its power, the Turing test methodology may be difficult to implement in practice: (1) it requires more expert time, and ideally these experts should not have been involved in the development of the original system, (2) comparison against multiple experts can be difficult to measure, and (3) blinding the outputs from computer and human expert, which normally has to be done by putting them into a common form, can be very time consuming. For these reasons, a Turing test is normally a 'one-off' procedure near the end of development just prior to release.

6.2.3. *Simulation*

In some instances, an analogy to case testing is connecting the system to a simulation model. Each simulation run is a 'test case' and different scenarios with various parameter settings can produce a number of different runs (Hall and Heinze 1989). For simple deterministic simulation models, validation via simulation is very powerful. For complex domains, however, a major problem arises: the simulation is itself a model, not perfect, that performs within an acceptable range. Hence modeling problems, such as accuracy and reliability, cascade. An ES that performs well with a simulation can not be guaranteed to perform as well with the real system.

Where simulation comes into its own is with real-time control ES. Here it is important to validate the performance of the ES over time, and typically control of the real system will be not possible until the ES has proved performance, or in any case will not generate example results quickly enough. Sets of decisions made when controlling a simulation run can be mapped against time and the simulation output parameters, and compared to expert decisions or investigated for consistency and accuracy. Radwan *et al.* (1989) present a verification of a traffic signal control system called SCII that makes extensive use of a simulation model.

6.2.4. *Control groups*

Many ESs rely upon the combination of human user and system to solve problems, and hence the system can not be validated alone. Where this is the case, a Turing test can be combined with a control group methodology. Cases are presented to two separate groups: those with the system, and those without. The validation process then proceeds as before, although now it is anticipated that the group with the system out performs the control group.

This approach, unfortunately, contains many pit-falls. The two groups may have performed differently irrespective of one group having access to the system, and a small number of case studies may not show up this inherent difference. Complexities in the system may mean that performance with it may only improve over time, or that its effect on performance is negligible until fully institutionalized. Hence a control group approach is normally seen as an

evaluation tool used after implementation. For an example of using a control group as an evaluation tool see Hamilton and Chervany (1981); for an extensive discussion of evaluation using control groups, and other so called quasi-experiments, see Adelman (1991).

6.2.5. *Sensitivity analysis*

Where no case studies are available, the validation process is far more difficult. Often developers will verify the system, and then simply use credibility as a complete surrogate measure: is the system credible to the expert, the developers, and the potential users? Yet a few validation methods are applicable where no or few case studies exist, in particular *sensitivity analysis.*

Assume there exists a single case C where intermediate results, the line of reasoning and the final results are all known to be perfect (or, more likely, are judged by an expert to be reasonable). If C uses inputs $i_1, i_2 \ldots, i_n$, then each can be systematically altered (either individually, or in sensible combinations), and the change in output from the system assessed as reasonable or otherwise by an expert. Graphs can be drawn to depict the input/output relationship. In many instances, it is easy to generate tests where the output should *not* change given changes in input. For example, if in a financial analysis ES it is known that a particular figure should have no effect on the results (perhaps since the situation is dominated by other concerns), then the system should be run with this figure set at its extreme values.

One major pit-fall with sensitivity analysis is that starting with a few cases and altering them is unlikely to cover a large part of the input domain. Working from the attributes in the knowledge base, it may be more useful to generate synthetic test cases that give good coverage, for example using Miller's generic test method (Miller 1990), and then perform sensitivity analysis with variations of these.

6.2.6. *Comparison against other models*

In some instances a different type of model, such as an optimization or statistical model, may already exist. Comparison of the system against this model can provide useful insights, for example, Moninger *et al.* (1988) compare a weather forecasting ES against a regression model to assess the comparative accuracy of the system. Typically it might be expected that a knowledge-based approach will be able to handle odd or different cases better due to use of specific knowledge.

The availability of induction algorithms in many shells, such as Quinlan's ID3 (1979), means that induction of a rule set be quite easy to do. Although the induced set may be very limited compared to the rule set crafted by hand following knowledge acquisition, due to lack of examples or limitations in the algorithm used, it does provide an alternative that can give insights into the validity of the acquired set.

6.2.7. *Line of reasoning*

When validating many systems it is also necessary to show that the line of

reasoning is correct. There are two reasons for this: (1) if the user investigates the line of reasoning via facilities built into the shell, then it must be credible, otherwise the result will be disregarded, and (2) if the system is to be developed further than the reasoning process must be capable of being 'scaled up' to a larger domain (Chandrasekaran 1983). Line-of-reasoning can be used as evidence in a Turing test, however, this requires that human experts articulate their reasoning and that it can be presented to third-party experts in a form similar to the explanation facilities of the shell being used.

A more complex approach is to compare aspects of the reasoning process, such as the relative time taken to reason, the amount of data used, or the number of hypotheses established and rejected. Meservy *et al.* (1986) derived knowledge from experts using protocol analysis, and then the expert's percentage of time spent performing specific processes, such as 'cognitive processes' (e.g., assuming, conjecturing, evaluating and questioning) was compared to that of the system.

6.3. *Statistical methods*

Although many validations will be entirely qualitative in nature, in other instances a quantitative approach using a statistical model is warranted. Such an approach will almost certainly be necessary when comparing the system to one or more human experts.

Essentially, the validation process can be viewed as the following hypothesis test (O'Keefe *et al.* 1987):

> H_0: the ES is valid for the acceptable performance range under the prescribed input domain

where the alternative hypothesis is that the system is invalid. Hence, until the acceptable performance range has been established, and the type of problems the system will handle defined (and thus the future input domain prescribed) then statistical tests can not be formally employed. This test has been criticized by Hilden and Habbema (1990), who suggest that the onus should be on the developer to disprove the alternative hypothesis, i.e. prove the system is good rather than assume that 'the system is good as long as not proven bad'. However things are phrased, use of statistics should be preceded by an attempt to formalize what is being tested.

This section gives something of the flavor of the use of statistics. In any given situation, one particular approach from the vast array of statistical techniques may be better than others; Mosteller and Rourke (1973) covers many methods. If detailed statistic analysis is used, the validation team should include a statistician: misuse of statistical measures can be worse than no analysis.

6.3.1. *Continuous results*

Where a system produces a single result on a continuous scale (for example, a certainty factor representing an estimate of the financial state of a company),

then comparison against an expert estimate is quite straightforward. If the system's result is X_i, and that of the human expert is Y_i, then the difference between them will be $D_i = X_i - Y_i$. For n case studies, there will be n observed differences D_1 to D_n. The confidence interval for the difference between system and expert is thus:

$$d - t_{n-1,\,\alpha/2}\, Sd/\sqrt{n},\ d + t_{n-1,\,\alpha/2}\, Sd/\sqrt{n}$$

where d is the mean difference, Sd the standard deviation, and $t_{n-1,\,\alpha/2}$ the value from the t distribution with n degrees of freedom. If zero lies within the interval, then there is no significant difference between the system and the expert. Note that the acceptable performance range will dictate the specification of α.

A typical problem with this method is that a small number of case studies will generally give rise to a large confidence interval half-width, due to the small number of degrees of freedom, so any conclusions have to be carefully interpreted. Further, it is assumed that D_i is reasonably normally distributed.

Where a single result is not produced, this method can still be used if the output can be assessed as a whole. For example, in a Turing test, if the output of both system and expert are assessed by a third-party expert on a continuous scale of 1 to 10, then X_i and Y_i will be the absolute assessed performance measure. Where multiple results are produced, simultaneously applying a paired t-test to each result is inappropriate since the results may be correlated. O'Keefe *et al.* (1987) give an example of the correlated multiple response problem: 'in a medical diagnosis system prescribing drug treatment ... two types of drug can be validly prescribed if each is separately considered as independent — yet that combination of drugs may be unacceptable'. Hence it is necessary to produce *simultaneous confidence intervals*. How this can be done is shown in Balci and Sargent (1984).

6.3.2. *Categorical results*

Many ES, particularly classification systems, produce categorical results. These can be attributes, such as the name of a drug or tax form, or categorical measures, such as 'good', 'poor', etc. In a Turing test, evaluators are likely to use a Lickert scale or similar technique to assess a solution.

Originally developed to determine the consistency between raters using categorical scales, such as legal judges, *consistency measures* prove a useful tool when the system and known results or an expert are considered as two independent raters. Consistency measures are covered in detail in Fleiss (1981) and the discussion here uses the same notation.

The most useful consistency measure is the *kappa statistic*, originally developed by Cohen (1960), which measures agreement on a single category. The weighted kappa (Cohen 1968) measures overall agreement across all categories, and was used by Hickham *et al.* (1985) in the ONCOCIN validation. The study in King and Phythian (1992) makes extensive use of it. To explain the use of the weighted kappa, Table 1 shows the agreement between an expert and a system on twenty case studies with three result categories called *A*, *B* and *C*. Each

Table 1. Agreement between system and expert on 20 cases with 3 categories.

	Expert			
System	*A*	*B*	*C*	Total
A	0.5	0.1	0.05	0.65
B	0.1	0.1	0.05	0.25
C	0.05	0	0.05	0.1
Total	0.65	0.2	0.15	1.0

entry shows agreement as a proportion of the total cases, for example, the expert and the system both classified 50% of the cases as *A*.

The weighted kappa k is defined as:

$$k = \frac{p_o - p_e}{1 - p_e}$$

where p_o is the overall proportion of observed agreement, and p_e is the overall proportion of chance expected agreement, i.e. the agreement that would be expected to occur at random. A value of +1 for k indicates perfect agreement, and a value of 0 indicates that agreement occurs no more than would be expected by chance; a value less than 0 obviously indicates disagreement.

p_o is the sum of the agreement proportions, so here

$$p_o = 0.5 + 0.1 + 0.05 = 0.65,$$

and p_e is the product of the total classifications by rater, thus

$$p_e = (0.65 \times 0.65) + (0.2 \times 0.25) + (0.15 \times 0.1) = 0.4875.$$

hence $k = 0.3171$, which suggests that agreement does not vary much from what it would be by chance.

This can be formally tested. Fleiss (1981) gives an expression for the standard error of the kappa, $s.e._o(k)$. This example gives $s.e._o(k) = 0.1962$, and thus a z statistic can be calculated as

$$z = \frac{k}{s.e._o(k)} = \frac{0.3171}{0.1962} = 1.6163$$

and compared against a standard normal distribution. This is not significant given that the z value for $\alpha = 0.05$ is 1.96.

6.3.3. *Large sample comparisons*

In a few instances, particularly when connecting an ES to a simulation model, it is possible to generate very large amounts of data. Although the above methods can be used, an alternative (less stringent) test is:

H_0: under the prescribed input domain, the distribution of results from the ES is equivalent to that of the comparative method

We typically can use a Chi-Square test to test for distribution equivalence, and this works particularly well for categorical data. However, even with a large sample, the Chi-Square is only valid if at least 5 data points are in each category. Norman and Naveed (1990) attempt to use this method to validate a cement kiln control ES, but then reduce their test to an ad-hoc comparison of proportions due to the lack of data points in some categories.

6.3.4. *Multiple experts*

For many ES, dealing with comparison against multiple experts is a major headache (Shaw and Woodward 1988). The weighted kappa can be used to measure agreement between n raters. Again using the notation of Fleiss (1981), if there are k categories and m ratings, the weighted kappa k is:

$$k = 1 - \frac{nm^2 - \sum_{i=1}^{n} \sum_{j=1}^{k} x_{ij}^2}{nm(m-1) \sum_{j=1}^{k} p_j(1-p_j)}.$$

Producing a kappa for multiple experts prior to validation of the system is a useful measure. If agreement between human experts is good, then the system can be compared against the joint agreement of all other experts using a statistic developed by Williams (1976). If they do not agree, decisions have to be made regarding validation criteria. Kappa comparisons across each pair of experts, and each expert and the system, can be used to indicate who agrees with whom.

7. THE MANAGEMENT OF VERIFICATION AND VALIDATION

As with all aspects of ES development, V&V must be appropriately managed. O'Keefe and O'Leary (forthcoming) state that managing V&V is more than *performing* the necessary tests — it is equally the process of *planning* what will be done where, and *acting* upon the results of any testing. Such actions may be difficult decisions, for instance delaying the implementation of a system until further development and testing and complete, or even terminating its development.

Management decisions regarding the process and outcomes of V&V will interact with the development process throughout the software life-cycle (Harrison 1989). Although many ESs are developed without an explicit development methodology, and hence their life-cycle is not well formed, fundamental development stages are likely to be identifiable, including knowledge acquisition, prototyping and implementation and maintenance (O'Keefe and Lee 1990). Some development models for ESs that specifically include V&V have been proposed, and three are very briefly reviewed here.

7.1. *Location in the life-cycle*

The position of validation in the life-cycle has been discussed in detail in Gaschnig *et al.* (1983). Recently, Benbasat and Dhaliwal (1989) have considered the location of methods in the life cycle as a substantial portion of the basis of the choice of validation methods.

There are a number of reasons why the life cycle has an impact on determining the type and extent of V&V. First, it is critical to verify the programmed knowledge base before substantial validation efforts are begun. If validators find errors in the knowledge base due to inappropriate implementation of particular technologies, then substantial doubt may be cast on the ability of the developers to develop a correct system. The credibility of the system may be damaged.

Second, some approaches to the validation of ESs compare portions of the knowledge base at different stages of the life cycle (for example, O'Leary and Kandelin 1988). The expected relationships between information is compared to the actual relationships, in order to gauge the extent of further validation efforts. Third, V&V efforts required in the initial stages of development are likely to be different than those required later in the process. Initial efforts may concentrate on direct examination of components of the knowledge, while later efforts are likely to be aimed at validation of the system as a whole.

7.1.1. *Knowledge acquisition*

There has been an increasing emphasis on validation during knowledge acquisition (Shaw and Woodward 1988; Enand *et al.* 1990), particularly when the acquisition processes is automated. An early commitment to verification in knowledge acquisition, as a preventive approach, can help knowledge engineers maintain the integrity of knowledge and reduce the iterative development cycle (Benbasat and Dhaliwal 1989). As with other software, catching problems early will save considerable effort downstream.

Boose (1986) and Gaines (1987) suggest that knowledge engineers prepare diagrams representing a summary of the problem solution process and give it to the experts for verification. These can be influence diagrams, activity graphs, decision tables, or classification hierarchies. The visual interaction can enhance mutual understanding of the problem and lead the domain expert to more structured reasoning.

Verification facilities in an automated tool can provide real time verification of expert knowledge: many automated knowledge acquisition tools now provide some method of automated verification, frequently a mechanism for checking for inconsistency or conflicts. For example, KNACK (Klinker *et al.* 1987) checks contradictions amongst incoming knowledge and eliminates ambiguities in input terminologies when acquiring knowledge from multiple sources, and ETS (Boose 1986) and ACQUINAS (Boose and Bradshaw 1987) maintain consistency by presenting knowledge in multiple forms to experts who can validate the knowledge.

7.1.2. *Prototyping*

Much of the validation effort can be done at the prototype stage, and need not be left until a full system is developed. As noted in O'Leary (1988c), prototypes can be an important validation tool, since ". . . prototypes provide an opportunity to test assumptions about the knowledge base, inference strategies of the expert and other characteristics of the system."

Incremental prototypes can serve as the source of requirements and enhance the developers' understanding of the system objectives and the users' expectations, as well as the system functionalities. Each prototype system, however, should be formally validated to gain the advantages of prototyping. O'Leary *et al.* (1990) present a methodology for formally validating a prototype. It combines face validation (where the prototype is assessed 'at face value') with sub-system (i.e., component) testing and system validation via case or Turing tests.

7.1.3. *Implementation and maintenance*

For systems that have a very low user's risk, it may be possible to 'field test' the system: put it in place, and let the users find errors. Cochran and Hutchins (1987) provide an insightful discussion of the problems of field testing based on experiences with a diagnosis system called MENTOR. Field testing is only possible if a user can determine when an error has occurred (for example, a piece of equipment still does not work after being mended under advice). For many systems, this will not be possible. When user's risk is high, a formal validation of the system prior to implementation is virtually obligatory (Hickam *et al.* 1985).

As an ES evolves over time there is a need to build the validation process into the maintenance of the system. This often means that it must be the responsibility of someone to validate new knowledge, alterations to existing knowledge, enhancements to the system, etc. In the O'Leary and Watkins (1991) survey, at least one company employed a knowledge base manager. That manager was responsible for a particular ES, ensuring that any new knowledge added to the system was verified and validated. Typically, a knowledge base manager will be someone with less technical understanding of the software but possibly more domain expertise.

ES maintenance systems have become increasingly important as systems grow in size and complexity. Such systems have varying capabilities, but verification is normally one of the primary concerns in order to ensure that the knowledge added to the system is consistent. Shatz *et al.* (1987) present one such system. Further, it is becoming evident that production of a system that is maintainable should be one of the major criteria for V&V, and this is particularly true for large ES (O'Neil and Glowinski 1990). Measures of rule base structure, such as in Jacob and Froscher (1991), and methods such as the RIME approach to engineering and modularizing XCON (Soloway *et al.* 1987) are thus of increasing importance.

If case studies have been previously used to validate a system, then they may be used to revalidate a system. Hence *regression testing*, checking that an ES can

still perform as previously observed after alteration, is important for many implemented systems. Often, unfortunately, this is not possible, since changes to knowledge, information, or organization operating procedures will make the cases outdated.

7.2. *The costs of validation*

Implicitly and explicitly, cost assessments permeate virtually all system development and validation efforts since there are always resource or time constraints. Hence the attention given to any part of the life-cycle is to some extent dependent upon the perceived benefit of validation in that part of the process.

The survey summarized in O'Leary (1991) found that validation efforts rarely exceed budget, and generally are allocated significantly less of the total system budget than is normally planned. Validation efforts are often driven out of the life-cycle by the production process of developing the ES, or perhaps are simply not done properly. Further development and enhancements are often perceived as more important than validation. Additionally, the costs and benefits of different validation methodologies are likely to depend on their location in the development process. For a system at the prototype level there may not be a detailed user interface, and as a result, it may prove quite costly to have the expert, end-user or sponsor involved directly in any validation efforts.

7.3. *Formality and standards*

Virtually all the factors discussed in this paper have an impact on the cost of validation of an ES. However, generally the formality of the validation effort will be the major factor in the determination of the cost of the validation, specifying who performs the validation, when the validation is performed and of what the validation consists.

In informal validation, the process is left to the developers and programmers, and possibly subsumed into other parts of the life-cycle. In formal validation, the cost of the validation will include the time of any experts (perhaps third-party), the time of the project sponsor, and the cost of acquiring or generating case studies or a simulation. A formal validation of the system is likely to take place at the conclusion of one of the major prototypes, and as a result, substantial up-front effort is likely to be made to ensure that the system meets the demands placed on it. Finally, in a formal validation, the process is likely to require a considerable amount of interaction with the system and include a specified sequence of tests, such as a Turing test.

In some organizations, it is likely that standardized and mandated tests will formalize the V&V process (Harrison and Ratcliffe 1991). This is commendable as long as the required tests are appropriate. The specter of inadequate but formal procedures driving necessary tests out of the development process exists.

7.4. *Life-cycle models*

If a formal development life-cycle is used, then V&V will have to fit into existing development stages, or new stages that focus on V&V will have to be designed. Weitzel and Kerschberg (1989) present a common solution to this problem: separate development stages representing testing and validation are added to the development model (in fact they split testing into two separate stages, one for reasoning and one for knowledge). This presents a problem if each, as is traditional with conventional systems, is located at the end of the cycle. The benefits of V&V that can be obtained by early commitment, discussed above, will not be obtained. In their Knowledge-Based System Development Life Cycle (KBSDLC), however, each stage is a *process* that can be activated as necessary, and hence testing and validation can be activated and performed at any time during development.

Miller (1989) presents a formal waterfall life-cycle, designed to fit into the reporting requirements of US Department of Defense projects. The process is driven by the system requirements from which an initial prototype is developed. Requirements are respecified following the evaluation of the prototype, and a system is developed and evaluated. This cycle repeats until the system is evaluated as ready for delivery. V&V occur at each system evaluation, and are more formally performed on the delivery system.

Whereas a formal waterfall method is often followed in projects where specific deadlines and deliverables must be met, and considerable documentation must be maintained, for many organizations a less formal approach will be more appropriate. O'Keefe and Lee (1990) present a variation of Boehm's spiral model. Requirements analysis is followed by knowledge acquisition and prototyping. After each knowledge acquisition stage the acceptable level of performance for the system can be specified, and this will become firmer and more detailed each time around the spiral. As a prototype evolves into a production system, verification (perhaps using mechanical automated methods) gives way to validation using test cases. When the acceptable level of performance is very clearly defined then statistical tests can be used.

8. SUMMARY AND CONCLUSIONS

ES V&V is less well developed than other aspects of the knowledge engineering process, particularly knowledge acquisition and knowledge representation. As more ESs have become developed and implemented, increasingly more attention has been given to all aspects of V&V, and as the development of ESs has become more formalized then the position of V&V in the life cycle has become important. Despite this, according to a recent survey of ES V&V practice (Hamilton *et al.* 1991), much V&V is ad-hoc and not closely integrated with development.

The different interpretations given to V&V, and also to assessment, credibility and evaluation, has made generalization difficult. Despite the efforts to

define and identify the important aspects of V&V (this paper included) different definitions and interpretations will likely continue. A developer using a shell to construct a system solely based upon expert knowledge will typically concentrate on verifying the knowledge and validating the performance of the system, using an overall approach more akin to validation in the model building disciplines than to software validation. Conversely, a developer working with a specification and a symbolic language to build a system only loosely based upon expert knowledge faces a software testing problem more akin to that found in traditional software development.

Table 2 summarizes the approaches to verification discussed in this paper. It should be remembered that software engineering approaches, not covered here, also plays a big role in many projects. The most popular approach is no doubt that centered on the concept of an anomaly — a potential error. For rule based systems, and some methods of handling uncertainty, a lot is known about

Table 2. Methods for ES verification, and examples of automated tools.

General approach	Specific approach	Example automated tools
Domain dependent	Meta-knowledge • represent meta-facts about domain, use these to detect inconsistencies in knowledge base	EVA, CLASP, various knowledge acquisition tools
Domain independent	Anomaly detection • detect and identify anomalies in knowledge base (rules and weights)	Table based: e.g., ESC, RCP Graph based: COVER TMS based: KB-REDUCER
	Statistical investigations • measure, e.g., attribute instances (static) and use of inference chains (dynamic)	
	Inference mechanism/conflict resolution • compare execution to specification or (if it exists) standard	

identifiable anomalies; for hybrid systems, let alone new approaches to development such as case based reasoning, a lot less is understood. Of the various tools developed to automatically detect anomalies, two stand out: COVER, the graph based domain independent tool, and EVA, which employs domain dependent meta-knowledge. Such tools are useful for large systems; for small ES with a limited rule set, perhaps than 200 rules, a cross reference checker such as CRSV and some common sense may be adequate.

Table 3 summarizes the approaches to validation discussed in the paper. The table shows the normal time that they can be used appropriately within the development life-cycle, and hence the approaches can be related to the life-cycle discussion. Many, if not most, ESs are validated using test cases, sometimes in the form of a Turing test where the performance of the system is compared against the performance of experts in a blinded evaluation. Statistical

Table 3. Methods for ES validation, and timing in the life-cycle.

General approach	Specific approach	Normal timing (early, middle, late)
Component	Rule validation • manually investigate important rules	Early
	Heuristic • compare performance to optimum • worst case analysis • distribution sampling	Middle
	Meta-models • construct and maintain conceptual model of the knowledge base	All
System	Case testing • ES solves cases, performance compared to expert	All
	Turing test • ES and experts solve cases, performances evaluated by third-party	Late
	Simulation • ES controls simulation model, performance evaluated	Late
	Control group • ES implemented for test group, performance compared to control group	Very late
	Sensitivity analysis • cases altered and input/output relationships evaluated	All
	Other model • another model, e.g. quantitative or induced, constructed and performance compared to ES	Late
	Line of reasoning • line of reasoning on test cases compared to elicitation material or expert	Middle

measures can aid the process. Component validation, which involves looking at specific rules and heuristics and perhaps constructing a meta-model of the knowledge base, can also be useful, particularly for large systems.

Advances in automation of all aspects of V&V, and their integration into every aspect of knowledge engineering, will continue to develop. Twenty years from now, the ES developer will likely use an electronic workbench that supports V&V of knowledge using meta-knowledge and structural checks, managers test cases, and provides supporting statistical tools. Despite this technology, there will no doubt still be ESs that make unforeseen and unpredicted errors.

ACKNOWLEDGEMENTS

This paper arose from tutorial presentations at the 1989 International Joint Conference on Artificial Intelligence and the 1991 and 1992 IEEE Conference on Artificial Intelligence Applications. We are grateful for the comments provided by Alun Preece (Concordia University, Montreal, Canada) and Sunro Lee (Rensselaer Polytechnic Institute, New York, U.S.A.).

REFERENCES

Adelman, L. (1991) 'Experiments, Quasi-experiments, and Case Studies: A Review of Empirical Methods for Evaluating Decision Support Systems', *IEEE Transactions on Systems, Man, and Cybernetics* **21**: 2, 293—301.

Adrion, W., Branstad, M. and Cherniavsky, J. (1982) 'Validation, Verification and Testing of Computer Software', *ACM Computing Surveys* **14**: 2, 159—192.

Agarwal, R. and Tanniru, M. (1992) 'A Petri-net Approach for Verifying the Integrity of Production Systems', *International Journal of Man-Machine Studies* **26**, 447—468.

Bachant, J. and McDermott, J. (1983) 'R1 Revisited: Four Years in the Trenches', *AI Magazine* **5**: 3, 21—32.

Balci, O. (1987) 'Credibility Assessment', in Balci, O. (ed.), *Proceedings of the 1987 Eastern Simulation Conference*, the Society for Computer Simulation, La Jolla, CA.

Balci, O. and Sargent, R. (1981) 'A Methodology for Cost Risk Analysis in the Statistical Validation of Simulation Models', *Communications of the ACM* **24**: 4, 190—197.

Balci, O. and Sargent, R. (1984) 'Validation of Simulation Models Via Simultaneous Confidence Intervals', *American Journal of Mathematics and Management Sciences* **4**: 3&4, 375—406.

Batarekh, A., Preece, A. D., Bennett, A. and Grogono, P. (1991) 'Specifying an Expert System', *Expert Systems with Applications* **2**, 285—303.

Bellman, K. L. (1990) 'The Modeling Issues Inherent in Testing and Evaluating Knowledge-based Systems', *Expert Systems With Applications* **1**: 3, 199—216.

Benbasat, I. and Dhaliwal, J. (1989) 'A Framework for the Validation of Knowledge Acquisition', *Knowledge Acquisition* **1**, 215—233.

Boehm, B. W. (1981) *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ.

Boose, J. and Bradshaw, J. (1987) 'Expertise Transfer and Complex Problems Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems', *International Journal of Man-Machine Systems* **26**, 3—28.

Boose, J. H. (1986) *Expertise Transfer for Expert System Design*, Elsevier, New York.

Bratko, I., Mozetic, I. and Lavrac, N. (1989) *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems*, MIT Press, Cambridge, MA.

Buchanan, B. and Shortliffe, E. (1985) *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Project*, Addison-Wesley, Reading, MA.

Buchanan, B., Sutherland, G. and Feigenbaum, E. A. (1969) 'Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry', in Michie, D. (ed.), *Machine Intelligence* **4**, Elsevier, NY.

Castore, G. (1987) 'Validation and Verification for Knowledge-based Control Systems', *Proceedings of the First Annual Workshop on Space Operations, Automation and Robotics*, NASA, pp. 197—202.

Chandrasekaran, B. (1983) 'On Evaluating AI Systems for Medical Diagnosis', *AI Magazine* **4**: 2, 34—37.

Chang, C. L., Combs, J. B. and Stachowitz, R. A. (1990) 'A Report on the Expert Systems Validation Associate (EVA)', *Expert systems With Applications* **1**: 3, 217—230.

Cochran, T. and Hutchins, B. (1987) 'Testing, Verifying and Releasing an Expert System: The

Case History of Mentor', *Proceedings of the Third IEEE Conference on AI Applications*, pp. 163—167.

Cohen, J. (1960) 'A Coefficient of Agreement for Nominal Scales', *Educational and Psychological Measurement* **20**, 37—46.

Cohen, J. (1968) 'Weighted Kappa: Nominal Scale Agreement with Provision for Scaled Disagreement or Partial Credit', *Psychological Bulletin* **70**: 4, 213—220.

Cragun, B. and Steudal, H. (1987) 'A Decision-table-based Processor for Checking Completeness and Consistency in Rule-based Expert Systems', *International Journal of Man-Machine Systems* **25**: 5, 633—648.

Davis, R. and Lenat, D. B. (1982) *Knowledge-based Systems in Artificial Intelligence*, McGraw-Hill, New York, NY.

Davis, R. (1984) 'Reasoning from First Principles in Electronic Troubleshooting', *International Journal of Man-Machine Studies* **24**, 347—410.

Duchessi, P., Shawky, H. and Seagle, J. P. (1988) 'A Knowledge-Engineered System for Commercial Loan Decisions', *Financial Management* **17**: 3, 57—65.

Duda, R., Gaschnig, J. and Hart, P. (1979) 'Model Design in the Prospector Consultant System for Mineral Exploration', in Michie, D. (ed.), *Expert Systems in the Microelectronic Age*, Edinburgh University Press, pp. 153—167.

Eglese, R. W. (1986) 'Heuristics in Operational Research', in Belton, V. and O'Keefe, R. M. (eds.), *Recent Developments in Operational Research*, Pergamon Press, Oxford, UK, pp. 49—68.

Enand, R., Kahn, G. S. and Mills, R. A. (1990) 'A Methodology for Validating Large Knowledge Bases', *International Journal of Man-Machine Studies* **33**, 361—371.

Ernst, C. J. (ed.) (1988) *Management Expert Systems*, Addison-Wesley, Reading, MA.

Fleiss, J. L. (1981) *Statistical Methods for Rates and Proportions*, John Wiley, NY.

Fox, M. S. (1990) 'AI and Expert System Myths, Legends, and Facts', *IEEE Expert* **5**: 1, 8—20.

Gains, B. R. (1987) 'An Overview of Knowledge Acquisition and Transfer', *International Journal of Man-Machine Studies* **26**, 453—472.

Gaschnig, J., Klahr, P., Pople, H., Shortliffe, E. and Terry, A. (1983) 'Evaluation of Expert Systems: Issues and Case Studies', in Hayes-Roth, F., Waterman, D. A. and Lenat, D. B. (eds.), *Building Expert Systems*, Addison-Wesley, Reading, MA, pp. 241—280.

Ginsberg, A. (1988) 'Knowledge-based Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy', *Proceedings of AAAI '88*, AAAI, Menlo Park, CA, pp. 585—589.

Ginsberg, A., Weiss, S. M. and Politakis, P. (1988) 'Automatic Knowledge Base Refinement for Classification Systems', *Artificial Intelligence* **35**, 197—226.

Gruhl, J. (1982) 'Model Credibility and Independent Evaluation: Three Case Studies', *Omega* **10**: 5, 525—537.

Hall, D. L. and Heinze, D. T. (1989) 'The Use of Simulation Techniques for Expert System Test and Evaluation', *ISA Transactions* **28**: 1, 19—22.

Hamilton, D., Kelley, K. and Culbert, C. (1991) 'State-of-the-practice in Knowledge-based System Verification and Validation', Technical Report, NASA/Johnson Space Center, Houston, TX.

Hamilton, S. and Chervany, N. L. (1981) 'Evaluating Information System Effectiveness — Part I: Comparing Evaluation Approaches', *MIS Quarterly* **5**: 3, 55—69.

Hansen, J. and Messier, W. (1986) 'A Preliminary Investigation of EDP-XPERT', *Auditing: A Journal of Theory and Practice* **6**: 1, 109—123.

Harrison, P. R. (1989) 'Testing and Evaluation of Knowledge-Based Systems', in Liebowitz, J. and De Salvo, D. A. (eds.), *Structuring Expert Systems*, Prentice-Hall, Englewood Cliffs, NJ, pp. 303—329.

Harrison, P. R. and Ratcliffe, P. A. (1991) 'Towards Standards for the Validation of Expert Systems', *Expert Systems With Applications* **2**, 251—258.

Hickam, D. H., Shortliffe, E. H., Bischoff, M. B., Scott, A. C. and Jacobs, C. D. (1985) 'The Treatment Advice of a Computer-based Cancer Chemotherapy Protocol Advisor', *Annals of Internal Medicine* **103**, 928—936.

Hilden, J. and Habbeman, J. D. F. (1990) 'Evaluation of Clinical Decision Aids — More to Think About', *Medical Informatics* 15: 3, 275—284.

Jackson, P. (1986) *Introduction to Expert Systems*, Addison-Wesley, Reading, MA.

Jacob, R. J. K. and Froscher, J. N. (1990) 'A Software Engineering Methodology for Rule-based Systems', *IEEE Transactions on Knowledge and Data Engineering* **2**: 2, 173—189.

Jafar, M. J. and Bahill, A. T. (1990) 'Validator, A Tool for Verifying and Validating Personal Computer Based Expert Systems', in Brown, D. E. and White C. C. (eds.), *Operations Research and Artificial Intelligence: The Integration of Problem Solving Strategies*, Kluwer Academic Press, Boston, MA.

Keen, P. W. (1981) 'Value Analysis: Justifying Decision Support Systems', *MIS Quarterly* **5**: 1, 1—15.

Kerlinger, F. (1973) *Foundations of Behavioral Research*, Holt, Reinhart & Winston, New York.

King, M. and Phythian, G. J. (1992) 'Validating an Expert Support System for Tender Enquiry Evaluation: A Case Study', *Journal of the Operational Research Society* **43**, 203—214.

Klinker, G., Bentolila, J., Genetet, S., Grimes, M. and McDermott, J. (1987) 'KNACK — Report-Driven Knowledge Acquisition', *International Journal of Man-Machine Studies* **26**, 65—79.

Kulikowski, C. A. and Weiss, S. H. (1982) 'Representation of Expert Knowledge for Consultation: the Casnet and Expert Projects', in Szolovits, P. (ed.), *Artificial Intelligence in Medicine*, Westview Press, Boulder, CO, pp. 21—56.

Laudaner, C. (1990) 'Correctness Principles for Rule-based Systems', *Expert Systems With Applications* **1**: 3, 291—316.

Landry, M., Malouin, J.-L. and Oral, M. (1983) 'Model Validation in Operations Research', *European Journal of Operational Research* **14**, 207—220.

Langlotz, C. P. and Shortliffe, E. H. (1983) 'Adapting a Consultation System to Critique User Plans', *International Journal of Man-Machine Studies* **19**, 479—496.

Langlotz, C. P., Shortliffe, E. H. and Fagan, L. M. (1986) 'Using Decision Theory to Justify Heuristics', in *Proceedings of AAAI '86*, AAAI, Menlo Park, CA, pp. 215—219.

Lee, S. and O'Keefe, R. M. 'Subsumption Anomalies in Hybrid Knowledge-bases', *International Journal of Expert Systems* (forthcoming).

Lehner, P. (1989) 'Toward an Empirical Approach to Evaluating the Knowledge Base of an Expert System', *IEEE Transactions on Systems, Man and Cybernetics* **19**: 3, 658—662.

Lethan, H. and Jacobsen, H. (1987) 'ESKORT — An Expert System for Auditing VAT Accounts', *Proceedings of Expert Systems and their Applications*, Avignon, France.

Liebowitz, J. (1986) 'Useful Approach for Evaluating *Expert Systems*', *Expert Systems* **2**: 3, 86—96.

Liu, N. K. and Dillon, T. (1991) 'An Approach Towards the Verification of Expert Systems Using Numerical Petri Nets', *International Journal of Intelligent Systems* **6**, 255—276.

Meservy, R., Bailey, A. and Johnson, P. (1986) 'Internal Control Evaluation: A Computational Model of the Review Process', *Auditing: A Journal of Theory and Practice* **6**: 1, 44—74.

Messier, W. F. and Hansen, J. V. (1992) 'A Case Study and Field Evaluation of EDP-XPERT', *International Journal of Intelligent Systems in Accounting, Finance and Management* **1**: 3, 173—186.

Miller, L. A. (1989) 'A Comprehensive Approach to the Verification and Validation of Knowledge-Based Systems', in *Proceedings of the 1989 AAAI Workshop on Verification, Validation and Testing of Knowledge-Based Systems*, AAAI, Menlo Park, CA.

Miller, L. A. (1990) 'Dynamic Testing of Knowledge Bases Using the Heuristic Testing Approach', *Expert Systems with Applications* **1**: 3, 249—269.

Moninger, W. R., Stewart, T. R. and McIntosh, P. (1988) 'Validation of Knowledge-Based Systems for Probabilistic Reasoning', in *Proceedings of the 1988 AAAI Workshop on Verification, Validation and Testing of Knowledge-Based Systems*, AAAI, Menlo Park, CA.

Mosteller, F. and Rourke, R. E. K. (1973) *Sturdy Statistics*, Addison Wesley, Reading, MA.

Nazareth, D. (1989) 'Issues in the Verification of Knowledge in Rule-Based Systems', *International Journal of Man-Machine Studies* **30**, 255—271.

Nguyen, T., Perkins, W., Laffery, T. and Pecora, D. (1985) 'Checking an Expert Systems

Knowledge Base for Consistency and Completeness', *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 374–378.

Nguyen, T., Perkins, W., Laffery, T. and Pecora, D. (1987) 'Knowledge Base Verification', *AI Magazine* **8**: 2, 65–79.

Norman, P. and Naveed, S. (1990) 'A Comparison of Expert System and Human Operator Performance for Cement Kiln Operation', *Journal of the Operational Research Society* **41**: 11, 1007–1019.

O'Keefe, R. M. (1989) 'The Evaluation of Decision-aiding Systems: Guidelines and Methods', *Information and Management* **17**, 217–226.

O'Keefe, R. M. and Lee, S. (1990) 'An Integrative Model of Expert System Verification and Validation', *Expert Systems and Their Application* **1**: 3, 231–236.

O'Keefe, R. M. and O'Leary, D. E. 'Managing and Performing Expert System Validation', in Grabowski, M. and Wallace, W. A. (eds.), *Advances in Expert Systems and Artificial Intelligence for Management*, JAI Press (forthcoming).

O'Keefe, R. M., Balci, O. and Smith, E. (1987) 'Validating Expert System Performance', *IEEE Expert* **2**: 4, 81–89.

O'Leary, D. (1987) 'Validation of Expert Systems', *Decision Sciences* **18**: 3, 468–486.

O'Leary, D. (1988a) 'Methods of Validating Expert Systems', *Interfaces* **18**: 6, 72–79.

O'Leary, D. (1988b) 'On the Representation and the Impact of Reliability on Expert System Weights', *International Journal of Man-Machine Studies* **29**: 6, 637–646.

O'Leary, D. (1988c) 'Expert System Prototyping as a Research Tool', in Turban, E. and Watkins, P. (eds.), *Applied Expert Systems*, North-Holland, Amsterdam, pp. 17–32.

O'Leary, D. (1990a) 'Soliciting Weights or Probabilities from Experts for Rule-Based Systems', *International Journal of Man-Machine Studies* **32**, 293–301.

O'Leary, D. (1990b) 'Verification of Frames and Semantic Networks', in Gaines, B. (ed.), *Proceedings of the Fourth Annual Workshop on Knowledge Acquisition*, Banff, Canada.

O'Leary, D. (1991) 'Design, Development and Validation of Expert Systems: A Survey of Developers', in *Verification, Validation and Testing of Expert Systems*, John Wiley, New York, NY, pp. 3–19.

O'Leary, D. and Kandelin, N. (1988) 'Validating the Weights in Rule-based Expert Systems', *International Journal of Expert Systems* **1**: 3, 253–279.

O'Leary, D. and Watkins, P. (1989) *Expert Systems in Internal Auditing*, Research Monograph, Institute of Internal Auditors.

O'Leary, T. J., Goul, M., Moffitt, K. E. and Radwan, A. E. (1990) 'Validating Expert Systems', *IEEE Expert* **5**: 3, 51–58.

O'Neil, M. and Glowinski, A. (1990) 'Evaluating and Validating Very Large Knowledge-based Systems', *Medical Informatics* **15**: 3, 237–252.

Ow, P. and Smith, S. (1987) 'Two Design Principles for Knowledge-based Systems', *Decision Sciences* **18**: 3, 430–447.

Pearce, D. A. (1988) 'The Induction of Fault Diagnosis Systems from Qualitative Models', *Proceedings of AAAI '88*, AAAI, Menlo Park, CA, pp. 353–357.

Preece, A. D. (1989) 'Verification of Rule-based Systems in Wide Domains', in Shadbolt, N. (ed.), *Research and Development in Expert Systems VI*, Cambridge University Press, pp. 66–77.

Preece, A. D. (1990) 'Towards a Methodology for Evaluating Expert Systems', *Expert Systems* **7**: 4, 215–223.

Preece, A. D., Shinghal, R. and Batarekh, A. (1992) 'Verifying Expert Systems: A Logical Framework and a Practical Tool', *Expert Systems With Applications* **5**, 421–436.

Quinlan, J. R. (1979) 'Discovering Rules by Induction from Large Collections of Samples', in Michie, D. (ed.), *Expert Systems in the Microelectronic Age*, Edinburgh University Press, UK, pp. 168–201.

Radwan, A. E., Goul, M., O'Leary, T. J. and Moffitt, K. (1989) 'A Verification Approach for Knowledge-based Systems', *Transportation Research-A* **23A**: 4, 287–300.

Rushby, J. (1988) *Quality Measures and Assurance for AI Software*, NASA Contract Report 4187, Washington DC.

Shatz, H., Strahs, R. and Campbell, L. (1987) 'ExperTAX: The Issue of Long-Term Maintenance', *Proceedings of the 3rd International Conference on Expert Systems*, pp. 291—300.

Shaw, M. and Woodward, J. (1988) 'Validation in a Knowledge Support System: Construing and Consistency with Multiple Experts', *International Journal of Man-Machine Studies* **29**: 3, 329—350.

Shpilberg, D. and Graham, L. E. (1989) 'Developing ExperTAX: An Expert System for Corporate Tax Accrual and Planning', in Vasarhelyi, M. A. (ed.), *Artificial Intelligence in Accounting and Auditing*, Markus Weiner, New York, NY, pp. 343—372.

Soloway, E., Bachant, J. and Jensen, K. (1987) 'Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a Very Large Rule-base', in *Proceedings of AAAI '87*, AAAI, Menlo Park, CA.

Suen, C. Y., Grogono, P. D. and Shingahl, R. (1990) 'Verifying, Validating and Measuring the Performance of Expert Systems', *Expert Systems With Applications* **1**, pp. 93—102.

Suwa, M., Scott, A. and Shortliffe, E. (1982) 'Completeness and Consistency in Rule-Based Expert Systems', *AI Magazine* **3**: 4, 16—21 (see also Buchanan and Shortliffe (1985), Chapter 8).

Turing, A. M. (1950) 'Computing Machinery and Intelligence', *Mind* **59**.

Waterman, D. A. (1986) *A Guide to Expert Systems*, Addison-Wesley, Reading, MA.

Weiss, S. M. and Kulikowski, C. A. (1984) *A Practical Guide to Designing Expert Systems*, Rowman, and Allenhead.

Weitzel, J. R. and Kershberg, L. (1989) 'Developing Knowledge-Based Systems: Reorganizing the System Development Life-Cycle', *Communications of the ACM* **32**, 482—487.

Williams, G. (1976) 'Comparing the Joint Agreement of Several Raters with Another Rater', *Biometrics* **32**: 2, 619—627.

Wyatt, J. and Spiegelhalter, D. (1990) 'Evaluating Medical Expert Systems: What to Test and How?', *Medical Informatics* **15**: 3, 205—217.

Yager, R. R. and Larsen, H. L. (1991) 'On Discovering Potential Inconsistencies in Validating Uncertain Knowledge Bases by Reflecting on the Input', *IEEE Transactions on Systems, Man and Cybernetics* **21**: 4, 790—801.

Yen, J., Neches, R. and MacGregor, R. (1991) 'CLASP: Integrating Term Subsumption Systems and Production Systems', *IEEE Transactions on Knowledge and Data Engineering* **3**: 1, 25—31.

Yu., V., Buchanan, B., Shortliffe, E., Wraith, S., Davis, R., Scott, A. and Cohen, S. (1979a) 'Evaluating the Performance of a Computer-based Consultant', *Computer Programs in Biomedicine* **9**: 1, 95—102.

Yu., V., Fagan, L., Wraith, S., Clancey, W., Scott, A., Hanigan, J., Blum, R., Buchanan, B. and Cohen S. (1979b) 'Antimicrobial Selection by Computer', *Journal of the American Medical Association* **242**: 12, 1279—1282 (see also Buchanan and Shortliffe (1985), Chapter 31).

Zlatareva, N. P. (1992) 'Truth Maintenance Systems and Their Application for Verifying Expert System Knowledge Bases', *Artificial Intelligence Review* **6**: 1, 67—108.