Building the Right System Right

Evaluating V&V Methods in Knowledge Engineering

Alun Preece

University of Aberdeen, Computing Science Department
Aberdeen AB24 3UE, Scotland
Phone: +44 1224 272296; FAX: +44 1224 273422
Email: apreece@csd.abdn.ac.uk

Abstract

This paper argues that verification and validation (V&V) techniques are an essential part of the knowledge engineering process, because they offer the only way to judge the success (or otherwise) of a KBS development project. However, an examination of known studies on the effectiveness of existing KBS V&V techniques shows, rather worryingly, that the state of knowledge in this area is very restricted and sparse. A proposal is made to improve this situation, by systematically gathering data from a representative set of KBS projects and V&V techniques. Without such a study, it is argued that knowledge engineering will remain very much an art.

The Art of Knowledge Engineering

Knowledge-based systems (KBS) have proven to be an effective technology for solving many kinds of problem in business and industry. KBS succeed in solving problems where solutions are derived from the application of a substantial body of knowledge, rather than by the application of an imperative algorithm. In the 1980s, KBS technology was widely applied to solve stand-alone problems. Classic examples of the successful use of the technology were in diagnostic problem-solving (for example, in medicine or engineering), provision of advice (for example, in "help-desk" applications), and construction/configuration (for example, product manufacturing and transportation loading). In the 1990s, many organisations have identified their collective knowledge as their most important resource, and are applying KBS technology to capture and exploit these "knowledge assets" in a systematic manner (Liebowitz and Wilcox, 1997).

The characteristic feature of problem domains where KBS technology is suitable is that the problems are *ill-defined*: they are not amenable to solution by algorithmic means; instead, the knowledge in the knowledge base of the KBS is used in some way to search for a solution. Often, the domain is such that there can be no guarantee that a solution will be found, or that found solutions will be optimal. Many KBS offer a "best effort" solution, which is good enough when the application requirements permit this (that is, the system is not safety or mission-critical).

The literature on KBS requirements specification recommends that the requirements be divided into *minimum* and *desired* functionality (Rushby, 1990): *minimum requirements* will often dictate what a system must never do (for example, a vehicle loading application must never produce a configuration that is unbalanced to the point of being dangerous to vehicle operators), while *desired requirements* will attempt to specify the quality of solutions (for example, that at least 90% of the configurations produced by the vehicle loading application should be within 15% of optimal). In practice, desired requirements will be difficult to specify, due to the ill-defined nature of the problem to be solved (for example, in the vehicle loading application, it may be very difficult to determine what constitutes an "optimal solution" for the desired requirements) (Batarekh, Preece, Bennett and Grogono, 1990). This is unsurprising; from a software engineering point-of-view, given the fact that the problem is ill-defined, it follows that the user requirements will be ill-defined also.

Knowledge engineering can be viewed as a special instance of software engineering, where the overall development strategy typically must employ exploratory prototyping: the requirements will typically be ill-defined at the outset, and it will take some effort in acquiring knowledge, and building prototype models, before the requirements can become more clearly defined. The knowledge engineer will have the hardest task when the domain knowledge itself is not well-understood; for example, when the knowledge is locked up in the heads of human experts who are not able to articulate it clearly. It is not unusual for a knowledge engineer to face a situation in which the users will be unable to say what they really want, experts will be unable to say what they really know, and somehow a KBS must be built! Building KBS is something of an art.

The Importance of Validation and Verification

Validation and verification (V&V) comprise a set of techniques used in software engineering (and, therefore, in knowledge engineering) to evaluate the quality of software systems (including KBS). There is much confusion about the distinction between validation and verification, but the conventional view is that *verification* is the process of checking whether the software system meets the *specified* requirements of the users, while *validation* is the process of checking whether the software system meets the *actual* requirements of the users. Boehm memorably characterised the difference as follows (Boehm, 1984):

Verification is building the system right.

Validation is building the right system.

Verification can be viewed as a part of validation: it is unlikely that a system that is not "built right" to be the "right system". However, verification is unlikely to be the whole of validation, due to the difficulty of capturing specifying user requirements. As noted above, this is a particularly important distinction in knowledge engineering. Of course, the goal in software/knowledge engineering is to try to ensure that the system is both "built right" and the "right system"; that is, the goal is to build "the right system, right".

In software engineering, efforts have been made to formalise the development process so that user requirements may be stated as a fully-formal specification, from which it can be proven that the implemented software system meets the requirements. While formal methods are desirable - even essential - in some cases (notably safety and mission-critical systems), these methods are unsuitable in large classes of software applications:

- Where requirements are amenable to formal specification, it may be too difficult to create the specification within project time and budgetary constraints.
- There are many kinds of requirement that are not amenable to formal specification (for example, the "usability" of a graphical user interface).

The extent to which formal methods can be applied in knowledge engineering is debatable (Meseguer and Preece, 1995), but it is certainly unrealistic to expect formal verification to serve as the only V&V technique in a KBS development project, because it will rarely be possible to ensure that the formal specification is a complete and correct statement of the users' requirements. Therefore, KBS V&V will typically need to involve multiple techniques, including formal verification against formal specifications (where possible), and empirical validation (including running test cases and evaluating the system in the operational environment) (Preece, 1990).

Given that knowledge engineering is an inexact art, the most fundamental measures of the success of a KBS project would seem to be:

Did we get it right? That is, does it meet the users' actual requirements.

Can we keep it right? That is, is it sufficiently maintainable for anticipated future changes.

Can we do it again? That is, is the process repeatable to ensure success with future projects.

The final point refers to the *capability* of the knowledge engineers, and reflects the modern view of software quality being determined primarily by the quality of the development process (Preece, 1995).

While verification and validation are only part of the overall development process, they are extremely important because they are the only way to produce an answer to the first of the three questions above ("Did we get it right?"), and provide partial answers to the other two questions (V&V techniques assist in measuring maintainability, and a repeatable V&V capability is a prerequisite for success in knowledge engineering).

Consideration of the importance of V&V to successful knowledge engineering raises another question: how effective are the KBS V&V techniques in current use? Obviously, if the techniques are incomplete or unsound, then they cannot be trusted to provide measurement of software quality and project success. The goal of this paper is to reflect upon studies which have been done to assess the effectiveness of current KBS V&V techniques, and to:

- summarise what the studies tell us about the current state-of-the-practice in KBS V&V;
- identify ways to improve the state of knowledge engineers' own knowledge about available KBS V&V techniques.

Unlike most papers on KBS V&V, the objective here is not to propose new V&V techniques, but to determine what can be done with the existing techniques, and propose further ways of measuring the effectiveness of current (and future) V&V techniques.

Knowledge Engineering = Method + Measurement

The previous section emphasised the importance of V&V as measurement techniques for the knowledge engineering process. Knowledge engineering (and software engineering) can be seen as a combination of *methods* and *measurement*: the methods used in requirements specification, knowledge acquisition, system design, and system implementation result in the production of a series of *artifacts* (Preece, 1995), each of which is amenable to some form of measurement (either individually or in combination). V&V techniques provide the means of obtaining the measurements. The following artifacts are of particular importance in the KBS development process:

Requirements Specification The requirements specification document states the minimum and desired user requirements (as described in Section 1), typically in natural language (or, less usually, in some restricted or semi-structured natural language subset). A framework for KBS requirements specification is given by Batarekh et al (Batarekh, Preece, Bennett and Grogono, 1990). As a natural language document, the requirements specification is not amenable to analysis by V&V techniques - instead, it is used to establish the needs for V&V.

Conceptual Model The conceptual model describes the knowledge content of the KBS in terms of real-world entities and relations. This description is entirely independent of the ways in which the KBS may be designed or implemented: the idea is to allow the knowledge engineer to perform a knowledge-level (epistemological) analysis of the required system before making any design or implementation choices. The best-known framework for defining KBS conceptual models is KADS (Wielinga, Schreiber and Breuker, 1992), in which models may be initially defined using a semi-formal, largely diagrammatic representation, from which a refined, formal model can be derived. The conceptual model forms the basis of the design model.

Design Model The design model serves to "operationalise" the conceptual model into an executable KBS; it describes the required system in terms of computational entities: data structures, processes, and so forth. For example, the design model may specify that a particular conceptual task is to be performed by a backward-chaining search, or that a concept taxonomy is to be represented using a frame hierarchy. The KBS specification language DESIRE is particularly well-suited to the representation of design models (Brazier, Keplics, Jennings and Treur). The design model dictates the form of the implemented system.

Implemented System This is the final product of the development process: the KBS itself. Once the design issues have been explored in the design model, the system may be implemented in any programming language, although typically a special-purpose KBS language is used.

There are many V&V techniques that have been developed for use on KBS - Gupta (Gupta, 1990) and Ayel and Laurent (Ayel and Laurent, 1991) provide good entry-points to the KBS V&V literature. Five of the most common approaches are listed below.

Inspection According to a survey of developers of KBS in business applications, inspection is the most commonly-employed V&V technique (O'Leary, 1991). Arguably, it is also the least reliable, as it essentially involves nothing more than human proof-reading the text of the various artifacts. Typically, a domain expert is asked to check the statements in the knowledge base; since the formal languages used in the design model and implemented system will be unfamiliar to domain experts, this technique is better-suited to use with the semi-formal conceptual model (which will typically use a more "reader-friendly" graphical representation).

Static Verification Static verification consists of checking the knowledge base of the KBS for logical *anomalies*. Frameworks for anomalies in rule-based KBS have been well-explored, and software tools exist to detect them (Preece, Shinghal and Batarekh, 1992). The most commonly-identified anomalies - and the ones detected by most of the available tools - are *redundancy* and *conflict*. Redundancy occurs when a knowledge base contains logical statements that play no purpose in the problem-solving behaviour of the system; this typically indicates that the system is incomplete in some way. Conflict occurs when there are logical statements that are mutually inconsistent, and would therefore cause the system to exhibit erroneous behaviour. Anomalies may exist in any of the formal artifacts: the implemented system, the design model, and - if it is defined formally - the conceptual model.

Formal Proof Formal proof is a more thorough form of logical analysis of the (formal) artifacts in the development process than that provided by static verification. As described in Section 1, where requirements are amenable to formal specification, proof techniques can be employed to verify that the formal artifact meets the specified requirements. A review of opportunities to use formal methods in

knowledge engineering is provided by Meseguer and Preece (Meseguer and Preece, 1995). In practice, however, while there are many formal specification languages for KBS, there are few documented examples of the use of proof techniques to very user requirements.

Cross-Reference Verification When there exists descriptions of the KBS at different "levels", it is desirable to perform cross-checking between these, to ensure consistency and completeness. For example, we would expect the concepts that are specified as being required at the conceptual level to be realised in terms of concrete entities at the design level, and in terms of concrete data structures in the implemented system. Therefore, the most appropriate uses of cross-reference verification are to check correspondence between:

- conceptual model and design model;
- design model and implemented system.

Empirical Testing All software testing involves running the system with *test cases*, and analysing the results. The software testing literature distinguishes between function-based testing and structure-based testing. *Function-based testing* bases the selection of test cases upon the functional requirements of the system, without regard for how the system is implemented. The success of function-based testing is dependent upon the existence of a "representative" set of test cases. In *structure-based testing*, test cases are selected on the basis of which structural components of the system they are expected to exercise; the objective is to show that the system produces acceptable results for a set of test cases that exercise all structural components of the system. Testing can be applied only to the executable artifacts: typically only the implemented system.

Table 1 summarises the applicability of the various types of V&V technique to the KBS development artifacts. The table shows only potential applicability of techniques to artifacts. The really important questions go beyond this, to ask:

- How effective is each of the techniques listed in Table 1, to provide some measurement of the quality of the appropriate artifact(s)?
- What combination of V&V techniques work best to provide the most cost-effective assurance of high quality for each artifact?
- What V&V techniques work best with which *method* for creating the artifacts?

The last question acknowledges the fact that not all V&V techniques can be used with all methods. For example, static verification to detect logical anomalies can be applied to the implemented system only if the implementation is created using a suitable programming language (one for which logical anomalies can be defined). Similarly, formal proofs can be applied to the design model only if an appropriate proof theory exists for the modelling language.

The following section examines the available data to discover to what extent the above questions can be answered now.

KBS V&V: How Well Are We Doing?

Surprisingly few studies are known to have been performed to evaluate the effectiveness of KBS V&V techniques. This section examines the results of five studies:

1. A comparative evaluation of several KBS verification and testing techniques, conducted at the University of Minnesota, USA (referred to here as "the Minnesota study").
2. A study comparing the effectiveness of an automatic rule base verification tool with manual testing techniques, conducted at SRI, USA ("the SRI study").
3. An examination of the utility of an anomaly detection tool, conducted at Concordia University, Canada ("the Concordia study").

Artifact	*V&V techniques*
Conceptual model	Inspection, Static verification (if formalised), Cross-ref verification (against Design model)
Design model	Inspection, Static verification, Formal proof, Cross-ref verification (against Conceptual model, Implemented system)
Implemented system	Inspection, Static verification, Testing, Cross-ref verification (against Design model)

Table 1: Applicability of V&V techniques to KBS development artifacts.

4. A comparative evaluation of several KBS verification tools/techniques, conducted by SAIC, USA ("the SAIC study").
5. A comparative evaluation of KBS verification and testing techniques, conducted at the University of Savoie, France ("the Savoie study").

The Minnesota study

Kirani, Zualkernan and Tsai (Kirani, Zualkernan and Tsai, 1992) at the University of Minnesota, USA, report on the application of several V&V techniques to a sample KBS in the domain of VLSI manufacturing. With the exception of a simple static verification (anomaly detection) tool, all of the methods used were manual testing techniques. The KBS itself was a 41-rule production system based upon well-understood physical properties of semiconductors, into which a variety of plausible faults were seeded. Interestingly, efforts were made to introduce faults at several different phases in the development process: at specification time, at design time, and at implementation time. A summary of the results is presented in Table 2.

The results of the study showed that the manual testing techniques, though labour-intensive, were highly effective, while the static verification tool performed poorly in detecting the seeded faults. Unfortunately, the success of the manual testing techniques could be attributed to the fact that this KBS application was exhaustively testable - which is rarely the case for industrial-scale KBS applications. Furthermore, given that the anomaly detection tool employed was of only the most basic type (able to compare pairs of rules only for conflict and redundancy), it is unsurprising that it performed poorly. Therefore, this study does not provide clear evidence - positive or negative - for the utility of modern KBS verification tools. Moreover, the study did not consider the complementary effects of the tools: no data was provided on which faults were detected by more than one V&V technique.

The SRI Study

Rushby and Crow (Rushby and Crow, 1990) at SRI, USA, like the Minnesota study, compared manual testing techniques with a simple static verification tool. The application used was a 100-rule forward-chaining production system in an aerospace domain, but the structure of the system was largely "flat" and very simple. Faults were not seeded in this study - instead, actual faults were discovered in the real application! - so there was no way to control the results. While interesting, this study does not yield reliable evidence as to the effectiveness of the V&V techniques employed.

The Concordia Study

Preece and Shinghal (Preece and Shinghal, 1994) at Concordia University, Canada, examined the use of a particular static verification tool, COVER, on a variety of KBS in different domains. The anomalies detected by COVER are as follows:

Redundancy Redundancy occurs when a KBS contains components which can be removed without effecting any of the behaviour of the system. This includes logically subsumed rules (`if p and q then r`, `if p then r`) rules which cannot be fired in any real situation, and rules which do not infer any usable conclusion.

Conflict Conflict occurs when it is possible to derive incompatible information from valid input. Conflicting rules (`if p then q`, `if p then not q`) are the most typical case of conflict.

Circularity Circularity occurs when a chain of inference in a KB forms a cycle (`if p then q`, `if q then p`).

Deficiency Deficiency occurs when there are valid inputs to the KB for which no rules apply (`p` is a valid input but there is no rule with `p` in the antecedent).

	Development phase		
V&V method	*Specification*	*Design*	*Implementation*
Static verification	38%	27%	19%
Structure-based testing	54%	68%	74%
Function-based testing	75%	92%	62%

Table 2: Summary of results of the Minnesota study: percentage of faults found for each phase by each V&V method.

COVER was applied to the following KBS (all of these were independently-developed, real KBS applications, not "toy" systems):

- MMU FDIR: a fault diagnosis/repair KBS developed by NASA/Lockheed);
- TAPES: a "help desk" product recommendation system developed by an adhesive tape manufacturer;
- DISPLAN: a health care planning system developed by the UK Health Service;
- DMS1: a fault diagnosis/repair KBS developed by Bell Canada).

A summary of the anomalies found by COVER appears in Table 3; the table also gives a measure of the complexity of each application, in terms of the number of objects in each knowledge base (rules, frames, or the equivalent, depending on the actual implementation language employed).

COVER was shown to detect genuine and potentially-serious faults in each system to which it was applied (in contradiction to the negative results on the use of this technique in the Minnesota study). Unfortunately, the Concordia study did not compare the effectiveness of COVER with other kinds of V&V technique.

The SAIC Study

Miller, Hayes and Mirsky (Miller, Hayes and Mirsky, 1993) at SAIC, USA, performed a controlled experiment on two KBS built in the nuclear power domain. Faults were seeded in each system, and groups of KBS developers and domain experts attempted to locate the faults using three different V&V techniques: manual inspection, static verification using the VERITE tool (an enhanced version of COVER (Preece and Shinghal, 1994), and static verification using MetaCheck, a simulated tool based on a conceptual enhancement of VERITE. The VERITE tool and the MetaCheck pseudo-tool were shown to provide significant assistance to both the groups of KBS developers and domain experts in locating faults:

- Groups using a tool (either VERITE or MetaCheck) found almost twice as many faults as the groups who did not have a tool, in 18% less time, with half as many falsely-identified faults.
- Groups using VERITE found 59% of seeded faults correctly.
- Groups using MetaCheck found 69% of seeded faults correctly.

While providing good evidence for the utility of static verification tools, and confirming the unreliability of manual inspection, the SAIC study did not compare static verification with empirical testing techniques.

The Savoie Study

Preece, Talbot and Vignollet (Preece, Talbot and Vignollet, 1997) at the University of Savoie, France, performed a comparative study of three V&V tools:

- SACCO: a static verification tool performing redundancy and conflict detection;
- COCTO: a static verification tool performing deficiency detection;
- SYCOJET: a structure-based testing tool capable of generating test cases to provide a specified level of knowledge base test coverage.

SACCO and SYCOJET are described in detail by Ayel and Vignollet (Ayel and Vignollet, 1993).

Independently-created sets of plausible faults were seeded into three different "mutated" versions of a real (207 rule) KBS application in an aerospace fault diagnosis domain. Each of the three tools was run on each of the three mutated KBS, and the results were aggregated; in summary:

- In each mutated system, at least 61% of faults were found by the combined effect of the three tools.
- SACCO always found at least 35% of the seeded faults.

	KBS			
	MMU	*TAPES*	*DISPLAN*	*DMS1*
Redundancy	10	5	5	7
Conflict	-	4	40	10
Circularity	-	-	4	-
Deficiency	-	16	17	-
KB size (objects)	170	230	405	1060

Table 3: Summary of results of the Concordia study: number of anomalies of each type found in each KBS.

- COCTO always found at least 27% of the seeded faults.
- SYCOJET always lead to the discovery of at least 27% of the seeded faults (with a test coverage of up to 46% of the rules - a level chosen for reasons of computational efficiency).
- The three tools were shown to be complementary in effect: less than 29% of faults detected were found by more than one tool.

Arguably, this study provides the best evidence yet that a combination of V&V techniques should be employed in any KBS development project. It also provides some useful evidence on the sensitivity of the different KBS techniques to different sets of seeded faults; however, three mutated KBS is not sufficient to provide any statistical confidence.

Conclusions from the Studies

The overall conclusion from the studies is that the collective knowledge on the effectiveness of KBS V&V techniques is very limited. There is some evidence that different techniques have complementary effectiveness, and no technique has been shown to be so weak as to be not worth employing. However, the data that is available is sparse, being limited to a few instances of KBS and specific applications of tools or techniques. It is almost impossible to combine the results of the different studies, because they were run with different types of KBS (for example, the Minnesota study used a "toy" KBS that was exhaustively testable, while the Savoie study used a genuine KBS application that was computationally too costly to attempt exhaustive testing), different instances of V&V techniques (the static verifiers used in each of the five studies *all* have different capabilities!), and different assumptions (for example, while the types of errors seeded in the Minnesota, SAIC and Savoie studies are similar, there are subtle differences which make cross-comparison hard).

The sparse nature of the available data is also evidenced by the fact that there is no known data for the effectiveness of formal proofs or cross-reference verification. Moreover, none of the studies apply V&V techniques directly to any artifact except the implemented system, and the implemented systems are almost exclusively rule-based.

The following section considers what can be done to improve this situation.

KBS V&V: What Do We Need To Do?

Clearly, in order to improve the collective state of knowledge on the effectiveness of KBS V&V techniques, it is necessary to perform a considerably larger set of studies. In order to gather a sufficiently complete data set, the following process would need to be followed:

1. Create a sufficiently complete enumeration of the types of KBS requiring V&V. For each type of KBS, create instance artifacts at each stage of development (conceptual model, design model, and implementation), and *for each development method.* For example, instances of KBS with various distinct problem-solving methods would be required, and artifact instances would need to be created using different methods (and representation languages).
2. Define reference implementations for each V&V technique, either in the form of well-defined manual procedures, software tool specifications/implementations, or a combination of the two. Where necessary, variations on the V&V techniques will need to be defined for different representations used in the reference KBS artifacts produced in Step 1.
3. Define good fault models, based on observed error phenomena from actual experience in KBS projects.
4. Mutate the KBS artifacts from Step 1 using the fault models from Step 3 (ideally, this would be done automatically); then apply each of the V&V techniques defined in Step 2 to each mutated artifact; repeat for a statistically-significant set of mutated artifacts.

Such a study would be very ambitious but extremely valuable: it would provide conclusive evidence as to the effectiveness of each V&V technique for each type of KBS and development method, individually and in combination. Furthermore, it would support further research and development of KBS V&V techniques. Of course, such a study would be very difficult: Step 1 and Step 3 in particular are made hard by the fact that KBS technology is moving constantly forward: new kinds of KBS are always emerging - for example, witness the current interest in multiple-agent KBS (Fisher and Wooldridge, 1993) - and reliable information on actual error phenomena is had to come by (partly because knowledge engineers do not wish to advertise failures). It is worth noting, however, that the artifacts created in Step 1 would be of wider use that merely in a study of V&V techniques - they could facilitate complementary studies on the effectiveness of knowledge acquisition and design methods.

Conclusion and Perspective

This paper has argued that V&V techniques are an essential part of the knowledge engineering process, because they offer the only way to judge the success (or otherwise) of a KBS development project. Examination of known studies on the effectiveness of existing KBS V&V techniques has shown, worryingly, that the state of knowledge in this area is poor. A proposal has been made to improve this situation, by systematically gathering data from a representative set of KBS projects and V&V techniques. Without such a study, knowledge engineering will remain very much an art.

In conclusion, however, it should be noted that the state of knowledge in software engineering is hardly much better! In particular, little is known about the relative effectiveness of V&V techniques in object-oriented software development. Despite this lack of knowledge, a huge number of successful, robust software systems have been created; similarly, a huge number of successful, robust KBS have been developed without perfect knowledge of the effectiveness of the methods employed. Clearly, software engineers, and knowledge engineers, have considerable artistic ability.

References

Ayel, M. and Laurent, J-P., Eds. (1991). *Verification, Validation and Test of Knowledge-based Systems*. John Wiley & Sons, New York.

Ayel, M. and Vignollet, L. (1993). SYCOJET and SACCO, two tools for verifying expert systems. *International Journal of Expert Systems: Research and Applications*, Vol. 6(2).

Batarekh, A., Preece, A., Bennett. A., and Grogono, P. (1996). Specifying an expert system. *Expert Systems with Applications*, Vol. 2(4).

Boehm, B. (1984). Verifying and validating software requirements and design specifications. *IEEE Software*, Vol. 1(1).

Brazier, F., Keplics, B., Jennings, N. and Treur, J. (1997). DESIRE: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, Vol. 6(1).

Fisher, M. and Wooldridge, M. (1993). Specifying and verifying distributed artificial intelligent systems. In *Progress in Artificial Intelligence-Sixth Portugese Conference on Artificial Intelligence (LNAI Volume 727)*, pages 13-28, Springer-Verlag, Berlin.

Gupta, U.G. (1990). *Validating and Verifying Knowledge-based Systems*. IEEE Press, Los Alamitos, CA.

Kirani, S., Zualkernan, I.A., and Tsai, W.T. (1992). *Comparative Evaluation of Expert System Testing Methods.* Technical report TR 92-30, Computer Science Department, University of Minnesota, Minneapolis.

Liebowitz, J., and Wilcox, L. (1997). *Knowledge Management and Its Integrative Elements*. CRC Press, New York.

Meseguer, P. and Preece, A. (1995). Verification and Validation of Knowledge-Based Systems with Formal Specifications. *Knowledge Engineering Review*, Vol. 10(4).

Miller, L., Hayes, J., and Mirsky, S. (1993). *Evaluation of Knowledge Base Certification Methods*. SAIC Report for U.S. Nuclear Regulatory Commission and Electrical Power Research Institute NUREG/CR-6316 SAIC-95/1028 Vol. 4.

O'Leary, D.E. (1991). Design, development and validation of expert systems: a survey of developers. In (Ayel and Laurent, 1991), pages 3-20.

Preece, A. (1990). Towards a methodology for evaluating expert systems. *Expert Systems*, Vol. 7(4).

Preece, A., Shinghal, R. and Batarekh, A. (1992). Principles and practice in verifying rule-based systems. *Knowledge Engineering Review*, Vol. 7(2).

Preece, A. (1995). Towards a Quality Assessment Framework for Knowledge-Based Systems. *Journal of Systems and Software*, Vol. 29(3).

Preece, A. and Shinghal, R. (1994). Foundation and application of knowledge base verification. *International Journal of Intelligent Systems*, Vol. 9(8).

Preece, A., Talbot, S. and Vignollet, L. (1997). Evaluation of Verification Tools for Knowledge-Based Systems. *International Journal of Human-Computer Studies*, Vol. 47.

Rushby, J. (1990). Validation and testing of Knowledge-Based Systems: how bad can it get? In (Gupta, 1990).

Rushby, J. and J Crow, J. (1990). *Evaluation of an Expert System for Fault Detection, Isolation, and Recovery in the Manned Maneuvering Unit*. NASA Contractor Report CR-187466, SRI International, Menlo Park CA.

Wielinga, B.J., Schreiber, A.Th., and Breuker, J.A. (1992). KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition*, Vol. 4(1).