Validation and verification of intelligent systems – what are they and how are they different?

Avelino J. Gonzalez
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL

Valerie Barr
Computer Science Department
Hofstra University
Hempstead, NY

Abstract

Researchers and practitioners in the field of expert systems all generally agree that to be useful, any fielded intelligent system must be adequately verified and validated. But what does this mean in concrete terms? What exactly is verification? What exactly is validation? How are they different? Many authors have attempted to define these terms and, as a result, several interpretations have surfaced. It is our opinion that there is great confusion as to what these terms mean, how they are different, and how they are implemented. This paper, therefore, has two aims – to clarify the meaning of the terms validation and verification as they apply to intelligent systems, and to describe how several researchers are implementing these. The second part of the paper, therefore, details some techniques that can be used to perform the verification and validation of systems. Also discussed is the role of testing as part of the above-mentioned processes.

1. Background

It seems that no article on the evaluation of intelligent systems can be written without first defining what validation and verification (V&V) mean. Many authors choose to define them by citing some of the more popular (but unclear) definitions, which we shall see later. Others realize the lack of clarity and choose to come up with their own definitions. Moreover, other authors throw up their hands and simply coin new terms. As a result, there is great confusion about exactly what validation and verification are, and how they can be implemented for intelligent systems.

intermingled and therefore, indistinguishable', as one author has suggested. (Smith 199x, p2) Nevertheless, we still believe that they are different, distinct, and individually valuable. Our focus, then, will be to shed some light and better understand what they mean, while, hopefully, not confusing the issue further. We begin by taking a tour through the relevant literature to see what other authors say about these terms.

1.1 Mainstream definitions

We begin by citing the IEEE (IEEE 1990), which defines validation and verification for conventional software as follows:

> **Verification**. (1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness.
>
> **Validation**. The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

Based on the above, it appears that the only difference between the two terms is that verification is done at the end of each development phase, while validation is done at the end of the development process. However, according to the above definition, validation can also be done 'during' the development process, which implies that it subsumes verification. Another slight difference in terminology is the formality of the processes -- verification is '… satisfy the conditions…' vs. '… satisfies specified requirements'. Nevertheless, all but the pickiest of readers will wonder what the difference is between the two as per the above definitions. So, the first 'official' definition proves to be of no help.

Another 'official' definition is given by the Department of Defense as it applies to simulation models. (DoDD 199x) It defines them as follows:

> Verification is the process of determining that a model implementation accurately represents the developer's conceptual descriptions and specifications
>
> Validation is the process of determining the degree to which a model is an accurate representation of the real world *from the perspective of intended uses of the model* (italics are theirs).

This definition certainly seems to differentiate between the two. It implies verification is the comparison with specification, while validation is a comparison with the real world. This is contradictory to the IEEE's definition, which states that validation is in fact comparison to specifications. It also implies that specifications are not necessarily representative of the real world (with which, by the way, we agree).

Intelligent systems are models of real-world practice, as are computer simulation systems. But these two types of systems are very different from each other. Similarly, we hope that good software engineering practice is used to develop intelligent systems, but the process typically used to do so is often quite different than that used to build conventional systems. Therefore, while we need to take the above definitions seriously, we recognize that they may not be completely appropriate for the development of intelligent systems. In any case, let's turn to what authors are saying about V&V specifically in the context of intelligent systems.

The definitions most often cited in the literature (by unofficial count) are by O'Keefe et al. (O'Keefe at al. 1987). The authors of that paper agree that validation is very often confused with verification, and proceed to try to clarify the issues with their definitions. They state that '*…verification refers to building the system "right" (that is, substantiating that a system correctly implements its specifications)*
validation refers to building the right system (that is, substantiating the system performs with an acceptable level of accuracy)'. While the definition of verification makes considerable sense, especially in the clarification within the parenthesis, the one for validation does not, in spite of the play on words employed. It is rather general, and does not address what the benchmarks should be. Furthermore, it does not explain what constitutes a 'right system'. What is accuracy? In relation to what? However, it is interesting that their definition seems to take the side of the DoDD definitions, which state that it is verification that compares the system with the specification, and not validation, as does the IEEE definition. In the defense of the authors, they state that they are mostly concerned with 'performance validation', and later in the article they proceed to answer the questions posed above in quite a reasonable manner. However, it is their definition that we still have trouble understanding.

Adrion et al. (Adrion et al. 1982) developed an early definition of the terms, albeit for conventional computer software. The authors define verification as '*…the demonstration of the consistency, completeness and correctness of the software*', and validation as '*… the determination of the correctness of the final program or software … with respect to the user needs and requirements*
incorporates the terms consistency and completeness, something not done by any of the other definers that we have seen here. We shall become very familiar with these terms as we go, however. But there is no mention of meeting specification requirements with respect to verification, but rather with validation.

Another source is a tutorial (O'Keefe and O'Leary 1993) where they use O'Keefe's 1987 definition. The interesting aspect of this is that they introduce a plethora of additional terminology such as 'assessment',

'credibility', 'evaluation' and 'system quality'. These new terms, unfortunately, tend to further confuse the issue. We will ignore them in our treatment.

1.2 Other definitions

Other authors also take a stab at defining these terms. Let's review some of these, and then see whether any patterns emerge:

Preece et al, 1992: '*Verification is the process of ensuring that the knowledge base of an expert system is free from internal errors …*'. '*Validation is the process of ensuring that the external behavior of an expert system conforms to specified requirements*'. Interesting concepts introduced by this definition: only the knowledge base is to be verified, and validation refers to external behaviour, presumably, as reflected by a set of testing.

Polat and Guvenir, 1991: '*Knowledge base verification, a part of the validation process, includes checking the knowledge base for completeness and consistency to guard against a variety of errors that can arise during the process of transferring human expertise to a computer system*'. Here we see the concept that verification is part of validation.

Chang, Combs, and Stachowitz, 1990: These authors do not explicitly define validation and verification. However, they appear to infer that validation is the process of checking for inconsistent, redundant, incomplete or incorrect properties in the knowledge base. This contradicts the definitions of other authors.

Zlatareva, 1992: '*Verification consists of detecting and correcting structural errors, and tests for completeness and violated semantic constraints*'. Validation, on the other hand, is '*…intended to assure the functional correctness of the system's performance*

Green and Keyes, 1987: Verification is determining the *traceability* of the system to its specification. They state that this is largely a paper exercise. Validation is defined as '*…the process by which the deliverable code is directly shown to satisfy … user requirements…*

Jafar, 1989: This doctoral dissertation repeats other definitions. It describes verification as (paraphrased) building the system right, and meeting specs. Also guarantees consistency of the system. Validation is building the right system; '*writing specifications and checking performance to make sure that the system does what it is supposed to do*'. The author also says that validation involves completeness and *conformance of output with established requirements*'.

Two authors independently compile different definitions of the terms. Knauf (Knauf 1999) introduces definitions from the field of psychology and of formal methods. He also brings in O'Keefe's definitions of the right system being built right. However, none of them add anything new to our discussion. McGraw and Harbison-Briggs (Harbison-Briggs 1989) also reproduce definitions from two other authors. They cite Jensen and Tonies (Jensen and Tonies 1979) who basically relate both to the specifications, without making a very good distinction. They also cite Glenford and Myers (Glenford and Myers 1976), who relate both to testing, again, without any real distinction as far as expert systems are concerned.

Closer to home, Barr (Barr 1996) defines verification as '*A static analysis process. Detects internal inconsistencies in a rule base such as redundancy, conflict or cycles (when not allowed). Check that the system is logically sound and complete*'. Validation is a '*… dynamic process. Determine that the system is behaving in accordance with specification. The conclusion of the system resembles that of the human expert who provided knowledge for the system*'.

This should be enough to demonstrate our point that confusion reigns. We now make some observations on the above sampling of definitions from the technical literature.

1.3 Observations

It should by now be obvious that we really don't know what we mean by verification and validation of intelligent systems, at least from the standpoint of the technical community. Some say that validation and verification are one and the same thing; others say verification deals with specs; others say it is validation that deals with specs; yet others say that both do. Furthermore, some authors relate consistency and completeness to verification while others do so with validation. Nevertheless, some trends do emerge. These trends will form the basis for our definition, which (we certainly hope!) will clarify and not further confuse the issue. These trends are not universally accepted by all authors, but rather are simply trends that we have noticed.

1. Trend #1: Verification deals with satisfying specifications.
2. Trend #2: Verification involves the structural correctness of the knowledge base. This is interpreted to mean that it attempts to ensure the internal consistency and completeness of the knowledge base.
3. Trend #3: Validation involves some type of exercising of the system – testing. A dynamic process; functional correctness.
4. Trend #4: Validation compares the system to the real world.

We will next use these trends to form new definitions that we hope will satisfy most readers.

2.0 Yet another definition of validation and verification

Before we embark on the definitions, there are some issues we must discuss:

2.1 The individual value of validation and verification

The first issue to consider is whether we should just forget our attempt to differentiate between the two processes and define a new process, such as 'assessment' or 'qualification', and leave it at that. As ones who like simplicity, that is indeed very tempting. But we strongly believe that there are in fact two separate and independent processes, and that there is inherent value in performing the two processes in sequence (first verification, then validation). Hopefully, the discussions herein will convince the reader of that. Furthermore, from a practical standpoint, the terms are so embedded in the lexicon that they would be very difficult to eradicate. Introducing new terms would simply add more confusion to the situation, not less.

2.2 Purpose of V&V

Another issue we confront is what should be the final purpose of V&V. Should it be designed to eliminate errors? Or should it serve as a means to merely certify that an intelligent system is free of errors? There are two things to consider here.

First, elimination of errors is akin to the old debugging process of the early days of programming. The system was exercised to discover an incorrect operation, and then the bug that caused any resulting fault could be identified and removed. This is a very useful and necessary thing to do, not just with computer programs but also with just about anything designed for gainful use. So, it is our opinion that V&V must include this aspect if it is to be useful.

Second, there is significant value in being able to say that a system is free of errors, and works as intended. Unfortunately, this is more wishful thinking than anything else. To proclaim a system to be free of errors is logically impossible unless a truly exhaustive way to inspect its functionality can be implemented. This is unfeasible for all but the most trivial of expert systems. Some concessions have to be made. These will be covered in our discussion of ways to implement validation.

As a result, we punt, and decide that V&V should be both a means to reduce errors (some authors refer to it as 'refinement') as well as a means to certify system 'correctness', however that may be specified. We

2.3 Human performance as a benchmark

When all the formal definitions are stripped away and we have argued endlessly about what V&V are, the core issue in validation and verification of an intelligent system boils down to one simple objective: ensuring the resulting system will provide an answer, solution or behaviour equivalent to what an expert in the field would say if given the same inputs.

But why use human expertise as the benchmark? Are there better or more appropriate benchmarks against which to determine equivalency? We shall now attempt to convince the reader that the benchmark of an intelligent system should, in fact, be human performance.

Basically, computers can be thought of as modeling infrastructures. They can be used to simulate many physical systems – either naturally occurring or man-made. Simulation of turbulent flow of air across the wings of an aircraft can replace the need for testing a prototype in a wind tunnel. Electrical circuits have been simulated very effectively for years, as have nuclear reactions, electromagnetic fields, hurricanes and other complex natural phenomena. Certainly computers can be other things (e.g. data archives, communication tools, control tools) but only the first description really impacts our discussion.

There have been many attempts to explain artificial intelligence. Most of these tend to fall back on a definition of human intelligence – something that has yet to be done successfully in our opinion. But we digress. From the perspective of the computer being a modeling machine, however, artificial intelligence is nothing more than simulating the workings of the human brain in a computer. This model can be defined as a white box (simulating the actual neurological operation of the brain), or as a black box (concerned only with modeling the input/output mappings).

There exist many domains that cannot be directly modeled, either because they are not well understood, or because they are too complex to model deterministically. But humans, as a result of their experience with such domains, are able to model these domains using some less-than-perfect heuristics or other techniques that permit them to work within that domain in an advantageous fashion. But the truth is that in most cases, if we could do so effectively and efficiently, we would probably choose to model the domain directly. (Knauf 1998) Therefore, intelligent systems represent models of domains that are indeed difficult or impossible to model other than through human intervention. More specifically, expert systems do not directly model the domain itself, but rather model the human's interpretation of the domain. Therefore, it logically follows that it should be human performance that serves as the benchmark for determining an expert system's 'correctness', rather than the domain itself, as the latter is typically impossible to model directly. This is a key to our definition of validation, as we shall see later.

We are now ready to define (or at least describe) what validation and verification should be.

2.4 Verification

Trends No. 1 and 2 both refer to verification. Based on them, we define verification as follows:

> **Verification is the process of ensuring that the intelligent system 1) conforms to specifications, and 2) its knowledge base is consistent and complete within itself.**

The intent of this definition is that the process of verification represents an internal benchmark, rather than an external one. Making it internal is highly significant, as errors can be found without the need to exercise the system with test cases. This makes the verification process very amenable to automation. In fact, this has been one area of great achievement by intelligent systems researchers. Now let's discuss what a specification should be, and what consistency and completeness should be.

2.4.1 Specifications

Specifications have traditionally been documents prepared by the purchaser or user of a device or system to describe what they want to buy. They include the device's maximum or minimum size, weight, speed,

capacity, pressure, voltage, power, materials of construction, operating conditions, user profile, etc. They can either be very detailed and constrictive, or they can be loose and flexible. Engineers will then design the system to be built such that it meets (but does not exceed!) the specifications. However, in computer software specifications are something somewhat different. They not only include operational characteristics of the software system, but may also include how the system is to be built. They are not prepared by the purchaser only, but also by the developers as a way to define interface criteria among the many developers possibly involved in the project, whose individual products have to interface with one another's. The specifications then become a working document, rather than being only a strict set of operational benchmarks to satisfy.

But in intelligent systems these distinctions blur. If we agree that intelligent systems simulate human intelligence, then it is only necessary to point to one task (or set of tasks) performed by a human, and specify that it be replicated it in a computer. That may be an oversimplification, as there may be aspects of that task or tasks that are more important than others, and the interface with the user always needs to be very carefully specified. But in general, specifications in intelligent systems have been of secondary importance. Here we interpret specifications to mean a body of knowledge that has been elicited from an expert, either as part of the knowledge acquisition part of the project, or contained in a book written by the expert in question. This body of knowledge is in a tangible form (written document, electronic document, drawing, etc.).

Under this interpretation, the first part of the definition of verification refers to ensuring that the system has, in fact, incorporated that knowledge. Checking for consistency and completeness of the knowledge base can ensure this.

There is a second aspect to ensuring that the systems satisfy the specs as in the traditional sense of specs. However, that is largely a paper process - one in which we have little interest here.

2.4.2 Consistency and completeness

We define consistency and completeness to be a state in which the knowledge base is free of internal errors. Internal, again, is the key word. For example, suppose we make the assertion that the sun rises in the west and sets in the east. As long as we always follow that in our system, we are being consistent internally. Never mind that the expert had in fact said the exact opposite thing. For verification, that is not important – at least the way we have defined it.

Conversely, inconsistency can be described as the presence of errors such as conflicts, redundancies, circularity (a particularly insidious one), and subsumptions. Incompleteness can be embodied in unreachability, dead-end modules, unneeded elements, and missing links. These imply that there is something else missing that should be added. Automatically detecting these errors has been investigated very thoroughly, and several systems exist that can do this very well.

Note, however, that there is no mention of performance against the real world (human expertise). This is left for validation

2.5 Validation

Trends #3 and #4 are clearly directed towards validation. This gives us the entrée to define it as follows:

> **Validation is the process of ensuring that the output of the intelligent system is equivalent to those of human experts when given the same inputs**.

In validation a verified system is compared against the real world (or rather, the expert's perception of the real world). This implies that we somehow interrogate the intelligent system for a response to some type of input. Clearly, this means testing. The concept of equivalency is purposely left rather fuzzy as the requirements for it may vary with domains. For example (albeit a drastic one!), in a system that chooses

wines, equivalency may not be nearly as strictly defined as one that monitors and diagnoses nuclear reactors. Equivalency should, in fact, be defined in the customer specifications.

Thus, there are three aspects to the validation problem – 1) how to interrogate the intelligent system in such a way that its range of operation is well covered, 2) how to tell whether the responses it provides are individually 'equivalent', and 3) how to make a validity statement for the entire system. These will be covered in section 4 below.

It should be noted that human expertise need not be drawn solely from the brains of experts. It can be found in books, drawings and even in raw data. However, the experts can serve as a convenient way to express that knowledge.

Figure 1 graphically depicts the difference between validation and verification as we have defined it here.

Figure 1 – Graphical depiction of the difference between validation and verification

3. Methods used for verification

As mentioned before, one major advantage of verifying the internal consistency and completeness of the knowledge base is that it can be done without exercising the system, a process that lends itself to automation. Numerous approaches have been developed for automated verification of intelligent systems, particularly in the area of rule-based systems. In general, these approaches focus on determining consistency and completeness and identifying anomalies within the intelligent system. However, there is some recent verification work that checks that an intelligent system conforms to specifications in a formal sense.

One of the earliest automated verification systems was the static rule checking utility developed as part of the ONCOCIN system (Suwa et al. 1982). ONCOCIN's verification capabilities are divided into checking for inconsistencies (defined as conflict, redundancy, and subsumption) and incompleteness (defined as missing rules). An automated rule checker is used which displays potential errors, allowing the expert to make the ultimate decision about what are actual errors. ONCOCIN examines the rule set to determine which condition parameters are used to conclude a given action parameter (equivalent to the 'path-expression' approach of KB-Reducer (Ginsberg 1987, 1988a) discussed below). These parameter combinations are then used to determine conflicts, redundancies, and missing rules. A weakness of the ONCOCIN approach is that it is limited to identifying problems at the rule level. That is, it cannot identify problems that are the result of longer reasoning chains, such as redundancies resulting from several inference steps. Furthermore, it cannot determine if all the rules it tests are actually reachable in the context of the entire system. Finally, the overall complexity is such that the approach is only really useful if the set of parameters is small (Preece 1989).

Another example of automated verification via static analysis of rule-based systems is CHECK, which was developed (Nguyen et al. 1985, 1987) to verify the consistency and completeness of knowledge based systems (without uncertainty) built using LES, the Lockheed Expert Systems development environment. The algorithms used in CHECK could, presumably, be applied to rule-bases developed in other environments, but the CHECK program itself could not be. Operation of CHECK is based on the construction of a dependency chart, which shows the dependencies among rules and between rules and classes. CHECK identifies the following rule-base problems (or potential problems): consistency checks (conflict, redundancy, subsumption, unnecessary IF conditions, circular rule chains); completeness checks (unreferenced attribute values, illegal attribute values, unreachable conclusions, dead-end goals, and dead-end IF conditions). For identification of cycles in the rule base, CHECK uses a cyclic graph detection algorithm based on an adjacency matrix graph implementation. For identification of other problems in the rule-base, CHECK iterates over rules and clauses within rules. The complexity of CHECK appears to be

better than that of ONCOCIN, but the tool is not suitable for general use since it depends on the rules being set up in a particular fashion.

Next we consider ESC (Expert System Checker) (Cragun and Steudel 1987) that used decision-tables to check the completeness and consistency of a knowledge base. ESC looked for discrepancies, ambiguities (including conflict), redundancies, and missing rules. It created an automated mapping of rules to decision tables, with the conditions and consequents forming the table rows and the rule numbers forming the columns. Conflicts and redundancies were determined in quadratic time, followed by a completeness check. If there were no conflicts or redundancies, then the completeness check would be carried out in linear time. However, if conflicts or redundancies were found then the completeness check required exponential time.

KB-Reducer (Ginsberg 1987, 1988a) was designed to carry out knowledge base reduction. However, it also functions as a verification tool that can check rule-bases for inconsistency and redundancy, including contradictions and redundancies that are the result of inference chains, not just pairs of rules. The process carried out by KB-Reducer involves calculation of all possible logically independent and minimal sets of inputs under which the knowledge base will conclude each assertion. In order for the reduction process to work, the rules of the knowledge base must form an acyclic network under the **depends-on** relation, where a rule *r* **depends-on** a rule *r'* if *r'* must fire in order for *r* to be able to fire. KBR3 (Dahl and Williamson 1993), based on KB-Reducer, was developed to assist in the maintenance of large knowledge bases. KBR3 is based on an 'application-neutral' language into which the knowledge base must be translated before it can be processed.

COVER (COmpleteness VERifier) (Preece 1989, Preece and Shingal 1992, Grogono et al. 1993) is a tool for verification of rule-based systems that carries out seven verification checks: redundancy, conflict, subsumption, unsatisfiable conditions (rules which cannot be fired, missing values), dead-end rules, circularity and missing rules. The rules must either be written in or converted to a language based on first-order logic, and COVER must be given the set of final hypotheses (classes), as well as information about any semantic constraints. The first phase of testing involves checking single rules in order to find unsatisfiable conditions and dead-end rules. Then redundancy, conflict, and subsumption among rule pairs are detected by symbolic comparison of pairs of rules. While these checks are carried out a dependency graph of the rules is constructed, essentially following the **depends-on** relation used in KB-Reducer. Then a cyclic graph detection algorithm is used on the dependency graph in order to detect cyclic inference chains. Finally, inference chains are traced to find more general cases of ambivalence, circularity, redundancy, and missing rules.

The work carried out by Zhang and Nguyen (Zhang and Nguyen 1989) used a Pr/T net representation to do static analysis of a rule base. First a Pr/T net is constructed from the rule-base, including a *transition* for each rule, and *places* for each antecedent and consequent. Then there is an attempt to find various sub-net patterns in the Pr/T net, where the sub-net patterns are indicative of inconsistency and incompleteness in the net (and therefore in the rule-base). If matches for these patterns are found, then the particular sections of the rule-base are studied to see if, in fact, they involve an actual error in the rule-base. The conditions that can be found by this method are redundant rule pairs, subsumed rules, circular rules, conflicting rule pairs, dangling conditions, useless conditions, and isolated rules. However, this work does not take certainty factors into account, and it also does not handle negated information in the rule-base.

Valiente (Valiente 1993) presents a method for redundancy and subsumption detection based on a hypergraph representation of the rule-base. His premise is that hypergraphs provide a more compact representation for rule-bases than do graphs, and they also facilitate use of graph transformations based on graph grammars and algebraic graph transformation. His basic process involves translating the knowledge base into its hypergraph representation, detecting and removing all redundant and subsumed rules by use of graph productions, and then translating the resulting hypergraph back into a knowledge base. His method is restricted to antecedents with conjunctions, and any disjunctions in a rule must be rewritten using multiple rules.

Finally, VSE and VSE-II (Hutter et al. 2000) are general-purpose verification systems that show promise for a range of software systems, including intelligent systems. VSE can be viewed as a CASE tool that supports software development from the earliest stages through code generation. System developers use formal representations of system properties and system requirements to aid the development process, and VSE is invoked to prove the specified requirements hold on the knowledge represented in the system. The core of VSE is an interactive inductive theorem prover, which uses full first-order predicate logic based on abstract data types and syntactic components that represent temporal constraints. VSE has been used for development of systems that handle robot control within a nuclear power plant, a North Sea storm surge barrier, and smartcard operation. Current work on VSE-II will tailor it towards the special needs of particular application domains, such as e-commerce systems.

4. Methods used for validation

In order to determine whether or not the output of an intelligent system is equivalent to that of a human expert, we must interrogate the intelligent system with test data. Therefore, all validation systems are characterized by some mechanism that facilitates review of system results on test data. Perhaps the earliest dynamic analysis method for validation of an intelligent system was TEIRESIAS (Davis 1985), developed for use with the MYCIN system. TEIRESIAS allows the rule-base developer to find in the rule-base the errors that led to incorrect conclusions. TEIRESIAS is both a debugging tool and a knowledge acquisition tool, allowing the alteration, deletion or addition of rules in order to fix an error. TEIRESIAS shows the user the reasoning that was used by the system to reach a conclusion, and the user can then approve of the rules used or indicate that there was an error and make corrections.

There are two aspects of TEIRESIAS that diminish its utility for expert systems developers. First, using TEIRESIAS is an iterative process. Upon discovery of an error during a consultation, TEIRESIAS helps the user build new rules to fix the error, but it does not then automatically test the impact of the new rules on test cases which have already run successfully. The user has to rerun those test cases to ensure that the changes did not create new errors and did, in fact, fix the old errors. Secondly, effective use of TEIRESIAS depends on having a knowledge base tester who is sufficiently expert in the problem domain to know that there is an error in the reasoning used to reach a conclusion. This can limit its effectiveness in the usual expert systems development environment in which the expert provides information to the systems developer but is not necessarily actively engaged in the development and testing process.

The development of TEIRESIAS was followed by EMYCIN (van Melle et al. 1985), which fixes spelling errors, checks that rules are semantically and syntactically correct, and points out interactions among rules that could lead to errors. EMYCIN uses a trace of the system's reasoning process, an interactive mechanism for reviewing and correcting the system's conclusions (like TEIRESIAS), and a facility that compares the system's results with stored correct results for the test cases.

There are also a number of dynamic analysis tools that carry out (or assist) knowledge base refinement. Among these are SEEK (Politakis and Weiss 1984) and SEEK2 (Ginsberg et al. 1985, 1988a). We categorise these as validation systems, since rule refinements are carried out in order to enable a modified knowledge-based system to produce equivalent output. SEEK is an interactive rule refinement system that uses stored cases to provide guidance for rule modifications. SEEK makes suggestions of possible rule generalizations and specializations. A generalization weakens a rule so that the antecedent will be satisfied in additional situations, whereas rule specialization strengthens the rule so that the antecedent will be satisfied in fewer situations than originally. The user then decides which refinements to try out and which to incorporate into the rule-base. As each test case is run, SEEK matches the expert's conclusion for that case with the system's conclusion, creating a performance summary. Once the performance summary is complete the user can refine the rules, weakening or strengthening rules with the guidance of heuristics built into SEEK. SEEK2 improves on SEEK by automating many of the knowledge base refinement tasks. For example, SEEK2 automatically decides for which conclusions rules should be evaluated. It also decides which rule refinements to try, and which of those to keep in the final knowledge base, as indicated by improvements when the refined knowledge base is used to evaluate the test cases.

Path Hunter and Path Tracer (Grossner et al. 1993, Preece et al. 1993) are based on the idea that functional validation may show that the system performs well on the test cases, but there may still be problems in portions of the rule base that were never exercised during testing. The goal of Path Hunter/Path Tracer is the selection of a set of test cases that exercise the structural components of the rule-base as exhaustively as possible. This involves firing all rules, and also firing every 'causal sequence' of rules. The model used to identify all possible dynamic causal rule-firing sequences is the *rule execution path.*

Path Hunter is actually a verification tool, providing information about the system under test for subsequent use by Path Tracer. Through the process of applying Path Hunter to a rule-base, various rule-base anomalies can be identified, such as rule redundancies, isolated rules and ambiguous rules. Path Tracer is a tool for structural rule-base testing, using paths generated by Path Hunter, in conjunction with traces of dynamic rule firings, to determine how extensively the possible execution paths are covered by the test data.

A weakness of many validation approaches is that, because they focus on a set of test cases with known results, they only assess the system's behaviour for those known cases. However, this can lead to misleading prediction of the system's behaviour in actual use, and may not identify errors in the system because of lack of coverage by the test set. TRUBAC (Barr 1995, 1996, 1999a) is a tool for both verification and validation, based on the idea that functional testing alone is inadequate for demonstrating equivalent output, and that structural testing of an intelligent system (in this case, a rule-based system) must also be used. Following the lead of coverage-based approaches to testing procedural programs, TRUBAC uses a single graph representation (in contrast to COVER, which requires two representations in order to carry out verification alone). This representation can be used first to carry out a verification process that identifies redundancy, conflict, ambiguity, circular rules, dangling conditions and useless conclusions. After the rule-base is corrected to satisfy to verification criteria, then TRUBAC applies a series of coverage measures to assess the quality of the test set. In essence, TRUBAC allows us to both identify errors in the rule-base and identify incompleteness of the test set. TRUBAC can be used to enhance an incomplete test set based on a series of heuristics that use coverage information and meta-knowledge about the domain population (Barr 1997). An extension of TRUBAC (Barr 1999b) presents an approach for computing a safe prediction of a system's actual behaviour in real use by combining the coverage information provided by TRUBAC with information about the performance of the system on test data, meta-knowledge about the kinds of cases the system should handle and how likely they are in the population for which the system is designed, and information about how representative the test set is of the intended population. We can see that, where the majority of formal expert systems evaluation methods focus on verification only, or on a strictly functional approach to validation, by adding a structural analysis to the validation process the TRUBAC approach allows us to do both verification and validation as well improve the predictions we can make about future performance of the system.

Lastly, Knauf et al. (Knauf et al.1999) propose a complete methodology for the validation of rule-based intelligent systems. This methodology is composed of the following steps:

1. Test case generation – Like TRUBAC, their methodology attempts to generate a minimal set of test cases that provides sufficient coverage to meet the pre-defined validation criteria. It uses a combination of structural testing with combinations of inputs to generate the original test case set (called the Quasi-exhaustive set of test cases, or QuEST). Then it reasons about the input values as well as several validation criteria to cull the test case set to the smallest size possible for the requirements. This last set is called the Reasonable set of test cases (ReST)

2. Test case experimentation – The intelligent system being validated is interrogated via the set of test cases, and its response is recorded. The same test cases are given to a panel of experts (size and makeup not addressed) who likewise provide a set of responses to the test cases.

3. Test case evaluation – This phase involves a Turing-Test-like process through which the test case responses from the experts as well as the system are evaluated by the same panel of experts interrogated previously. The methodology provides several formulae that can be used

to identify the erroneous rule. The formulae take into account a judgment of the competency of each expert based on his/her responses.

4. Validity Assessment – This step of the process computes an assessment of the final overall validity of the system as presented. This validity is de-composed into output validity, rule validity, and test case validity. The rule validity is helpful in the next step.

5. System refinement – This step aims to improve the system, and makes use of the rule and test case validity to suggest how to improve the rule(s) found erroneous.

5. Summary and conclusions

We have surveyed the many definitions of validation and verification of intelligent systems, and have found them to be confusing at best, and contradictory at worst. In order to provide a uniform set of definitions for these terms, we have developed our own, based on an analysis of all the others. There are many application areas for intelligent systems, and there is much ongoing research on methods for verification and validation. However, it seems that there is a gulf between the theory of V&V and the actual use of V&V systems. This is similar to the situation in conventional software development, where there exist well-developed software design, development, and testing methodologies, but software developers often do not use them. In the case of intelligent systems, two obstacles have existed: the lack of a coherent set of definitions for verification and validation, and few tools that usable outside of the environment in which they have been developed. A uniform set of definitions should encourage developers to begin to think seriously about the need to perform formal V&V on their intelligent systems, and will also provide the foundation for researchers to develop tools that will be usable by others. The VSE system discussed in Hutter et al, [2000] is a good model in that the V&V tools are part of the development environment, facilitating V&V throughout the development process.

References

[Adrion et al, 1982] Adrion, W., Branstad, M. and Cherniavski, J., 1982, Validation, verification and testing of computer software. *IEEE Transactions on Systems, Man and Cybernetics*, **21** (2): 293-301.

[Barr 1995] Barr, V., 1995, TRUBAC: A tool for testing expert systems with rule-base coverage measures, In *Proceedings, 13th Annual Pacific Northwest Software Quality Conference*, Portland, OR.

[Barr, 1996] Barr, V., 1996, Applications of rule-base coverage measures to expert system evaluation, Rutgers University Technical Report DCS-TR-340.

[Barr 1997] Barr, V, 1997, Application of control flow and data flow analysis to rule-based systems. *Annals of Software Engineering*, **4**:171-189.

[Barr 1999a] Barr, V, 1999, Applications of rule-base coverage measures to expert system evaluation. *Journal of Knowledge Based Systems*, **12**:27-35.

[Barr 1999b] Barr, V., 1999, Applying reliability engineering to expert systems, In *Proceedings, FLAIRS-99*, Orlando, FL, pp. 494 - 498.

[Chang et al, 1990] Chang, C. L., Combs, J. B. and Stachowitz, R. A., 1990, A report of the Expert system Validation Associate (EVA). *Expert Systems with Applications*, **1**:217-230.

[Cragun and Steudel 1987] Cragun, B.J. and Steudel, H.J., 1987, A decision-table-based processor for checking completeness and consistency in rule-based expert systems. *International Journal of Man-Machine Studies*, **26:**633-648.

[Dahl and Williamson 1983] Dahl, M. and Williamson, K., 1983, Experiences of using verification tools for maintenance of rule-based systems. In *Working Notes, AAAI-93 Workshop on Validation and Verification of Knowledge-Based Systems*, Washington, D.C., pp. 114-119.

[Davis 1985] Davis, R., 1985, Interactive transfer of expertise. In B.G. Buchanan and E.H. Shortliffe (eds) *Rule-Based Expert Systems* (Reading MA: Addison Wesley), pp. 171-205.

[DoDD, 199x] US Department of Defense Directive DoDD 5000.59, 199x.

[Glenford and Myers, 1976] Glenford, J. and Myers, S., 1976, *Software Reliability – Principles and Practices* (New York: John Wiley and Sons).

[Ginsberg et al. 1985] Ginsberg, A., Weiss, S., and Politakis, P., 1985, SEEK2: A generalized approach to automatic knowledge base refinement. In *Proceedings, IJCAI-85*.

[Ginsberg 1987] Ginsberg, A., 1987, A new approach to checking knowledge bases for inconsistency and redundancy. In *Proceedings, Third Annual Expert Systems in Government Conference*, Washington, D.C., pp. 102-111.

[Ginsberg 1988a] Ginsberg, A., 1988, Knowledge-base reduction: a new approach to checking knowledge bases for inconsistency and redundancy. In *Proceedings, Seventh Annual National Conference on Artificial Intelligence*, pp. 585-589.

[Grogono et al. 1993] Grogono, P.D., Preece, A.D., Shingal, R. and Suen, C.Y., 1993, A review of expert systems evaluation techniques. In *Working Notes, AAAI-93 Workshop on Validation and Verification of Knowledge-Based Systems*, Washington, D.C., pp. 120-125.

[Green and Keyes, 1987] Green, C. J. R. and Keyes, M. M., 1987, Verification and validation of expert systems. In *Proceedings, Western Conference on Expert Systems*, Anaheim, CA, pp. 38-43.

[Grossner et al. 1993] Grossner, C., Preece, A. D., Chander, P.G., Radhakrishnan, T. and Suen, C.Y., 1993, Exploring the structure of rule based systems. In *Proceedings, AAAI-93*, Washington, D.C., pp. 704-709.

[Hutter et al. 2000] Hutter, D., Langenstein, B., Rock, G., Siekmann, J.H., Stephan, W., Vogt, R., 2000, Formal software development in Verification Support Environment (VSE). *Journal of Experimental and Theoretical AI*, this issue.

[IEEE, 1990] IEEE Std. 610.12-1990, Glossary of Software Engineering Terminology, 1990.

[Jafar, 1989] Jafar, M. J., 1989, A tool for interactive verification and validation of rule-based expert systems, Doctoral Dissertation, University of Arizona.

[Jensen and Tonies, 1979] Jensen, R. and Tonies, C. (eds), 1979, *Software Engineering* (Englewood Cliffs, NJ: Prentice Hall).

[Knauf, 1998] Knauf, R., 1998, Personal communication.

[Knauf, 1999] Knauf, R., 1999, Validating rule-based systems – a complete methodology. Habilitation Thesis, Technical University of Ilmenau, Germany.

[Knauf et al, 1999] Knauf, R., Gonzalez, A. J. and Jantke, K. P., 1999, Validating Rule-Based Systems: A Complete Methodology, In *Proceedings, IEEE Conference on Systems, Man and Cybernetics*, Tokyo, pp. V-744 - V-749.

[McGraw and Harbison-Briggs, 1989] McGraw, K. L. and Harbison-Briggs, K., 1989, *Knowledge Acquisition – Principles and Guidelines* (Englewood Cliffs, NJ: Prentice Hall).

[Nguyen et al. 1985] Nguyen, T.A., Perkins, W. A., Laffey, T. J., Pecora, D., 1985, Checking an expert systems knowledge base for consistency and completeness. In *Proceedings, IJCAI-85*, Menlo Park, CA, pp. 374-378.

[Nguyen et al. 1987] Nguyen, T.A., Perkins, W. A., Laffey, T. J., Pecora, D., 1987, Knowledge base verification. *AI Magazine*, **8** (2):69-75.

[O'Keefe et al. 1987] O'Keefe, R. M., Balci, O. and Smith, E. P., 1987, Validating expert systems performance. *IEEE Expert*, **Winter**:81.

[O'Keefe and O'Leary, 1993] O'Keefe, R. M. and O'Leary, D. E., 1993, Expert system verification and validation: a survey and tutorial. *Artificial Intelligence Review*, **7**:3-42.

[Politakis and Weiss 1984] Politakis, P. and Weiss, S., 1984, Using empirical analysis to refine expert system knowledge bases. *Artificial Intelligence*, **22**.

[Preece 1989] Preece, A.D., 1989, Verification of rule-based expert systems in wide domains. In *Research and development in Expert Systems VI, Proceedings of Expert Systems '89*, London, pp. 66-77.

[Preece and Shingal 1992] Preece, A. D., Shinghal, R., 1992, Analysis of verification methods for expert systems. In *Working Notes, AAAI-92 Workshop on Validation and Verification of Knowledge-Based Systems*, San Jose, CA.

[Preece et al. 1992] Preece, A. D., Shinghal, R. and Batarekh, A., 1992, Verifying expert systems: a logical framework and a practical tool. *Expert Systems with Applications*, **5**:pp. 421-436.

[Preece et al. 1993] Preece, A.D., Grossner, C., Chander, P.G., and Radhakrishnan, T., 1993, Structural validation of expert systems using a formal model. In *Working Notes, AAAI-93 Workshop on Validation and Verification of Knowledge-Based Systems*, Washington, D.C., pp. 19-26.

[Smith, 199x] Smith, Suzanne, "…..", Doctoral Dissertation, Florida State University, 199x.

[Suwa et al., 1982] Suwa, M., Scott, S. C., and Shortliffe, E. H., 1982, An approach to verifying completeness and consistency in rule-based expert systems. *AI Magazine*, **3** (4):16-21.

[Valiente 1993] Valiente, G., 1993, Verification of knowledge base redundancy and subsumption using graph transformation. *International Journal of Expert Systems*, **6** (3):41-355.

[van Melle et al. 1985] van Melle, W., Shortliffe, E.H., and Buchanan, B.G., 1985, EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems. In B.G. Buchanan and E.H. Shortliffe (eds) *Rule-Based Expert Systems* (Reading MA: Addison Wesley), pp 302-313.

[Zhang and Nguyen 1989] Zhang, D. and Nguyen, D., 1989, A technique for knowledge base verification. In *Proceedings, IEEE International Workshop on Tools for Artificial Intelligence*, pp. 399-406.

[Zlatareva, 1992] Zlatareva, N., 1992, A framework for knowledge-based system validation, verification and refinement: the VVR system, In *Proceedings, FLAIRS-92*, Ft. Lauderdale, FL, pp. 10-14.