Verification and Validation: A Case Study of a Decision Support System

A. Terry Bahill, K. Bharathan, and Richard F. Curlee,
Systems and Industrial Engineering
University of Arizona
Tucson, AZ 85721-0020
terry@sie.arizona.edu

Abstract

In 1985 we started building a decision support system to help speech clinicians diagnose small children who have begun to stutter. This paper describes the verification and validation of that system:
(1) having an expert use and evaluate it,
(2) running test cases,
(3) developing a program to detect redundant rules,
(4) using the Analytic Hierarchy Process,
(5) running a program that checks a knowledge base for consistency and completeness,
(6) having five experts independently critique the system,
(7) obtaining diagnoses of stuttering from these five experts derived from reports of children who had been evaluated for possible stuttering problems,
(8) using the system to expose missing and ambiguous information in 30 clinical reports, and
(9) analyzing the dispersion and bias of six experts and the decision support system in diagnosing stuttering. When using the final system, three clinicians with widely differing backgrounds produced diagnostic opinions that evidence little variability and were indistinguishable from those of a panel of five experienced clinicians.

I. Introduction

The development of a decision support system to help speech clinicians diagnose and manage preschool children at risk for a stuttering disability began in 1985. Several years of testing and modification . took place before *Childhood Stuttering: A Second Opinion*™ was commercially released in 1993.

Speech-language clinicians vary widely in their diagnoses of stuttering in young children, but discussing such cases with experienced clinicians greatly reduces this variability. It was found that using *Childhood Stuttering: A Second Opinion*, led clinicians with different training and experiences to diagnostic opinions indistinguishable from those of a panel of five experienced clinicians. In effect, using Second Opinion allows inexperienced clinicians to "discuss" cases of incipient stuttering with a panel of experts, a process that should increase the reliability of their diagnoses in the real world.

In the course of its development, the product went through several versions, each of which was tested at a differing level of rigor.

II. Testing the First System

The first system was tested in 1985 in a simple manner by having the expert use it and tell us how he felt about it, what seemed right or wrong. This method was unsatisfactory largely in terms of the time required of the expert. Each time a minor adjustment was made to the knowledge base, the expert had no choice but to run the system in its entirety. The need for a testing technique that used test cases was realized and implemented with the next version.

III. Testing the Second System

A second system named *Stutter* was developed in 1987. A series of test cases were used to assess *Stutter*. The expert ran the system and answered all the questions as they applied to a child he had evaluated for a suspected stuttering problem. After *Stutter* presented its diagnostic conclusions and advice, the intermediate knowledge was saved into a disk file, and a text editor used to remove values derived by the inference engine, leaving only the user's responses to the system's questions. A range of files were similarly generated and the sets of answers were saved under different names. *Stutter* was constructed to get its inputs from such files, and each file thus became a test case. The knowledge engineer did not have to always depend on the expert's time to check each change in the knowledge base.

Many test case were accumulated and used for testing over the life of the project. Whenever a rule was added or modified, each test case was loaded into the system and used to look for run-time errors. When the modification consisted of an additional question to be asked of the user, the test files had to be maintained by the expert having to provide the answers to every case where the question was fired. While the expert's knowledge could at times be obtained by a phone call, this was not always feasible. While on the surface the problem did not appear to be eliminated, in actual fact, the expert continued to maintain close contact with the evolution of the system, while his time was less taxed.

Other than the domain expert, knowledge engineers and end users are also good sources of test cases. Those created by a knowledge engineer are often more parsimonious because more rules can be exercised by fewer test cases. Typically, an expert, a knowledge engineer and an end user are like three blind men describing an elephant: each focuses on different aspects of the system. The test cases of each will exercise different but overlapping portions of the knowledge base; therefore it may be wise to always gather test cases from all three sources.

Next a computer program was created that used these test cases to test the expert advisory system more systematically (Kang & Bahill, 1990). The firing of each rule was recorded as each test case was run. A rule was said to succeed if all of its premises evaluated true and the action specified in its conclusion was performed. Rules that were never found to succeed for any test case were flagged as probable mistakes and were brought to the attention of the human expert. Of course, a rule that never succeeds during such testing may not be an error. Some rules may be intended for unusual or infrequently occurring cases that were not exercised by the test cases that were used. Conversely, rules that succeed for every test case may also be mistakes. If a rule is always true it may be better to replace it with a fact; however, there are control rules that must always succeed. Consequently, this technique is useful in screening a system's rules for potential errors; but a human must decide whether the rule in question is appropriate or not.

The best way to test an expert advisory system, in our opinion, is to have the domain expert run the system and create a representative sample of test cases. Next, determine which rules never succeed and which always do. Our experience indicates that a test case for every five rules in the knowledge base may be sufficient (i.e. 20 test cases for a 100 rule system) even though clearly more test cases are better and having many more test cases than rules would be best. For those rules that do not succeed, the expert should provide test cases that he thinks will make them succeed until most do succeed. Then, the knowledge engineer should try to determine the reason that the remaining rules are not succeeding. It should also be noted that some test cases are not real. The expert for our system characterized some test cases as cartoons that had idealized, exaggerated or stereotyped signs and symptoms of stuttering. To increase the number of test cases, he took a number of existing test cases and systematically changed those answers that should not change the system's output and looked to see if they did. Thus these types of test cases reflect an expert's conceptualization of a domain rather than real cases from the domain.

IV. Testing the Third System

The third expert advisory system, which was named Expert Stuttering Program (ESP), was ready for testing in 1989. It was the first to be tested with a special program, Validator, that we wrote to help find mistakes in knowledge bases (Jafar & Bahill, 1991; 1993). By this time, of course, there were many publications about verification and validation of particular knowledge-based systems, (referenced in Jafar & Bahill 1991).

Validator assesses the consistency and completeness of a knowledge base. It checks for syntactic errors, unused rules, unused facts, unused questions, incorrectly used legal values, redundant constructs, rules that use illegal values, and multiple methods for obtaining values for expressions and systematically indicates potential errors to the knowledge engineer.

A. Verification with Validator

Verification, or building the system right, ensures that a system correctly implements specifications and determines how well each prototype conforms to design requirements. It guarantees product consistency at the end of each phase (with itself and with previous prototypes) and ensures a smooth transition from one prototype to another.

Validator checks both the syntax and semantics of a knowledge base and brings potential errors to the attention of the knowledge engineer, who has the task of fixing such errors. Validator has four verification modules: a preprocessor, a syntax analyzer, a syntactic error checker, and a debugger.

The Preprocessor:

Syntactic errors can cause a production language (which is used here to refer to AI languages as well as expert-system shells) to misinterpret a knowledge base and consequently to alter its syntactic structure, leading to semantic errors. Validator's preprocessor performs a low-level syntax check. Detecting such errors saves knowledge engineers and experts much time, frustration, and grief. This type of checking is dependent on a system's production language and ameliorates the shortcomings of the language's compiler. Validator builds internal representation structures of the knowledge base, which its other three modules analyze for syntactic compliance.

The Syntax Analyzer:

Many syntactic errors are caused by misspellings, typographical errors, or ill-formed knowledge-base structures. Therefore, Validator's syntax analyzer creates alphabetical listings of the expressions in the knowledge base according to their categories: goals, rule premises, rule conclusions, questions (with legal values), and facts (with values). This list comprises a knowledge base dictionary that makes it easier for a knowledge engineer to find mistakes. For example, a knowledge engineer, forced to read an entire knowledge base, may be hard pressed to remember if a hyphen or an underscore is to be used to tie two words together. But if the two constructs are adjacent in a list it is easy to see an inconsistency.

The Syntactic Error Checker:

People can easily detect inconsistent expressions if they are presented in pairs rather than in large collections, but they need to rely on machines for detecting global and indirect inconsistencies. The syntactic error checker looks for syntax that, while legal, produces unspecified behavior by the production language's compiler. For example, "out-of-range values" (such as incorrect usage of reserved words) usually escape detection by the complier and the knowledge engineer. Validator detects these errors in the contexts in which they occur.

Expressions get their values from facts, from user responses to questions, or from the conclusions of rules. There are four basic types of values.

(1) Legal values are acceptable answers to questions.

(2) Utilized values appear in rule premises and allow the rule premise in which they appear to be evaluated to true.

(3) Concluded values appear in rule conclusions and are set for an expression when a rule using that expression succeeds.

(4) Assigned values are assigned to expressions with facts or certain commands specific to the production language.

(a) Utilized Versus Legal Values:

Legal values guard against typographical errors and help in abbreviating long answers. If no legal values are provided, a system accepts any response as a valid answer, so typographical errors and wrong responses may escape detection. Even if errors are detected, effective error-recovery procedures are time-consuming and increase the size of the knowledge base. It should be noted that providing legal values will not prevent a knowledge engineer from using out-of-range values in a rule. For example, Mycin-derived production languages do not check a knowledge base to ensure that only legal values have been used; these languages only check user responses to questions to see if they match the legal values specified by the knowledge engineer. Legal values are related to questions, not to rules or facts. Illegal values are common in knowledge bases and often result in the failure of rules using such values.

(b) Unused Legal Values:

The syntactic error checker searches the premises of rules, looking for declared but unused legal values, which it flags as potential errors. It also lists all unasked questions. Unused legal values are common in knowledge bases. Many result from errors, others are remnants of old constructs that were put into the knowledge base by mistake or were incompletely removed. Deleting such constructs reduces the size of the knowledge base and speeds inferences during the use of a system. If Validator searched the following knowledge base, it would note that the legal values for coat of animal are {hair, feathers, beads} but the utilized values are {hair, feathers, scales}. Validator would point out that it is not possible to get a value of beads for coat of animal and would indicate that this is likely a mistake. It would also point out that one legal value, scales, has never been used, and would suggest that this may also be a mistake.

```
goal = identity of animal.

if coat of animal = scales
then type of animal = fish.

if coat of animal = hair
then type of animal = mammal.

if coat of animal = feathers
then type of animal = bird.

if type of animal = lizard
then identity of animal = Gila monster.

question(coat of animal)=
 'What is the coat of animal?'.

legalvalues(coat of animal)=
[hair, feathers, beads].
```

(c) Utilized Versus Concluded Values:

If the values used in the premises of rules do not match the values used in the conclusions of those rules, the rules will fail. In the above knowledge base, the utilized values for type of animal are {fish, mammal, bird, lizard}, whereas the concluded values are {fish, mammal, bird}, which indicates that it would be impossible for this system to conclude that a lizard is a type of animal.

The Debugger:

Debugging is a tedious, difficult, time-consuming, and costly process of finding and correcting errors in the knowledge base. On many occasions, the presence of errors is discovered during developmental testing but finding them depends on the knowledge engineer's intuition, experience, and common sense. However, computer-aided debugging tools are more reliable because they reduce the possibility of human errors. Validator's debugger checks for rules that use variables, negations, and unknowns. It regards the knowledge base as a closed world, and assumes that all the information about the domain is captured in the knowledge base. Thus, all possible rules and axioms should be either implied or implicitly modeled in the knowledge base.

Variable-unsafe Rules: variables can be used in both the premises and the conclusions of rules. They act as symbolic place holders. Each construct that uses variables is logically equivalent to the large set of constructs that could be obtained by replacing those variables with suitable terms. A backward-chaining rule that has variables is variable-safe (closed) if:

(1) Its conclusion is homomorphic to a fact or a consequence in the knowledge base;

(2) Variables that appear as attributes of an expression in a conclusion also appear as attributes of an expression in the rule's premise;

(3) Variables that appear as attributes of an expression in a premise also appear as utilized values in previous premises of the same rule or as attributes of an expression in the conclusion;

(4) Variables that appear as concluded values of a rule also appear as utilized values in the rule's premise.

A knowledge base is variable-safe if its set of rules evaluate to true, no matter what values are substituted for its variables. Variable-unsafe rules usually lead to infinite loops that violate the closed-world assumptions.

Illegal Use of Negations: negations can be used only in the premises of rules. A rule that has a negation in its conclusion violates closed-world assumptions, because a false fact is not explicitly declared in the knowledge base, but is inferred from not being able to conclude a given value for an expression.

Illegal Use of Unknowns: the most common origin of unknown is the result of a user's response to a question. Unknown can also be concluded as a result of the unsuccessful firing of all the rules that concern an expression. However, like negations, unknown can be used only in the premises of rules. A rule that concludes unknown for an expression violates the closed-world assumption, because an unknown fact is not explicitly declared in the knowledge base; rather, it is inferred from not being able to conclude a value for an expression.

B. Validation with Validator

Validation, building the right system, ensures the consistency and completeness of the whole system. The validation part of Validator has two modules: a chaining thread tracer and a knowledge base completeness module. The chaining thread tracer determines if rules can fire by tracing their connectivity back to the goal. Rules that cannot fire are flagged as dead rules and are brought to the attention of the knowledge engineer. A rule is also flagged as dead if it is the root of a dead tree. Thus, flagging a dead rule may uncover a whole set of dead rules.

One aspect of validation is checking the knowledge base for completeness, that is, attempting to determine if something is missing from the knowledge base. This checking usually proceeds in two ways. The first is direct and involves a knowledge engineer and an expert working together reviewing, refining and analyzing each item in the knowledge base. They check the completeness of every module. They also check the consistency, effectiveness and efficiency of every knowledge base item. Then they review each rule and decide whether to split it, modify it, or delete it from the knowledge base.

The second approach requires a knowledge engineer to work on the structure and representation of the knowledge base. Knowledge base items are analyzed, compared for redundancy, completeness, and correctness of usage. This approach is structured, algorithmic, and more exhaustive than the direct approach. It also uses heuristics that can be automated to produce fast and effective results. Validator allowed us to use this second approach. After all potential errors flagged by Validator had been resolved, we continued testing this expert advisory system using test cases and additional domain experts.

V. Testing the Fourth System

The fourth version of this evolving decision support system, *Second Opinion*®, was ready for field testing in 1991. It had already undergone testing with all the techniques mentioned above when our knowledge engineer traveled to the universities of four other experts and spent a day with each running the system on real cases. The experts were generally pleased with the performance of the system, but each offered suggestions for changes, which were implemented. Next, the clinical records of 30 anonymous children whose parents suspected them of stuttering were collected and rewritten after deleting any information that might identify the child or his family. Each expert, at a workshop in Tucson, provided a diagnosis for each of the 30 children after reading the rewritten clinical reports. Next, they discussed these rewritten reports and their diagnosis. The latter diagnoses were used to derive a consensus diagnosis for each child and Second Opinion was changed to match the consensus opinions. Unfortunately, the performance of the system was not evaluated quantitatively before its modification.

If there is a collection of input-output data that is known to be correct (sometimes called a gold standard), then a variety of quantitative techniques can be used to help validate a system, many of these are described by (Adelman, 1992). In effect, the expert panel's diagnoses reflected our effort to develop such a gold standard.

VI. Testing the Fifth System

While testing a fifth version of this decision support system, *Childhood Stuttering: A Second Opinion*™ in 1992, we discovered that its outputs differed when different experts used it. We asked two experts to use the system based on information each obtained from reading the thirty clinical reports. The output of the system was the same for ten of these reports even though there were different answers to, on average, one-third of the questions. This suggests that the system is robust to differences attributable to users. The output of the system was highly similar for another ten reports but differed significantly on the remaining ten. These findings suggested that some differences might be attributable to the clinical reports. For example, one report stated "Jose felt frustrated." Some experts understood this statement to mean that the child was showing frustration about his disfluency, while others understood it to reflect his frustration with English and Spanish word-finding difficulties. It was decided, therefore, to resolve such ambiguities in these clinical reports before testing the decision support system further.

First, fifteen of the thirty clinical reports were carefully edited so that any lacunae, ambiguities, and conflicting statements were removed. Next, ten additional clinical reports were obtained and edited in a similar manner. These twenty five edited reports were then used to test the latest version of the decision support system. It would have been better, of course, to use a few hundred validated clinical reports, but developing these clinical reports was expensive. In 1992 and 1993 more time was spent acquiring and preparing these reports than in developing and refining the rules in the system's knowledge base.

A. New Tools

In Table I, diagnoses of four apocryphal experts and that of a decision support system for four clinical reports are displayed. These data are not real but were created to illustrate the use of several statistical tools. Diagnoses of the experts are denoted with lower case letters a, b, c, and d, and that of the decision support system with dss. The four clinical reports are designated by the upper case letters A, B, C, and D

Table I
Heuristic Data Set

Diagnostic	Name of Clinical Report			
Score	A	B	C	D
5			b	c, d
4			d	dss
3		b, d	c, dss	a
2	b, d, dss	dss	a	b
1	a, c	a, c		

The numbers in the Diagnostic Score column represent five diagnostic opinions, which correspond to the following descriptions:

1. Little cause for concern about stuttering. There is little reason to suspect that a stuttering problem may be emerging in this child at this time. The types and frequencies of disfluencies that were observed and described are characteristic of those of nonstuttering children.
2. Some cause for concern about stuttering. This child is evidencing some danger signs of incipient stuttering as well as some that are characteristic of nonstuttering children. This pattern of equivocal findings suggests that the child may be at risk for an emerging stuttering problem.
3. Mild concern about stuttering. This child is evidencing relatively consistent signs of early stuttering.
4. Moderate concern about stuttering. This child presents speech and behavioral signs which suggest that stuttering may not be a transient problem.
5. Severe concern about stuttering. This child evidences speech and behavioral signs that may signal the evolution of a severe stuttering problem.

Two quantitative measures of the agreement between the experts and the decision support system, which were originally suggested by Lucien Duckstein, can be used to characterize the dispersion and bias of these diagnostic data. Each diagnosis is denoted with a capital R with a subscript identifying the individual expert. The term dispersion shows how much the diagnosis of each expert varied from those of every other expert. It is computed with

$$Dispersion_k = \frac{1}{J}\sum_{j=A}^{D}\left(\sqrt{\frac{1}{I-1}\sum_{i=a}^{dss}(R_k - R_i)^2}\right)_j$$

where i runs over the set $\{a, b, c, d, dss\}$, k runs over the set $\{a, b, c, d, dss\}$, j runs over the set $\{A, B, C, D\}$, $I = 5$ (the number of evaluators) and $J = 4$ (the number of clinical reports).

For the data in Table I the dispersions are

"Expert"	Dispersion
a	1.47
b	1.68
c	1.37
d	1.35
dss	1.06

The term bias shows inclinations of the individuals. It is computed with

$$Bias_k = \frac{1}{J}\sum_{j=A}^{D}\left(\frac{1}{I-1}\sum_{i=a}^{dss}(R_k - R_i)\right)_j$$

For the data in Table II the biases are

"Expert"	Bias
a	-1.19
b	0.38
c	-0.25
d	1.00
dss	0.06

These measures can be used to help validate a decision support system. For the data in Table I all the "experts" performed less consistently than did the decision support system, because four experts had dispersion scores greater than that of the decision support system. The opinions of expert "b", for example, were the least consistent. The data also illustrate that experts may be biased. The diagnosis of expert "a" suggest that he viewed each child's fluency problem as less severe than the other experts, while those of expert "d" were at the other extreme. The decision support system, on the other hand, showed little bias. These data were contrived to illustrate such ideal behavior by a decision support system.

B. The Final System Test

The 25 new clinical reports were mailed to five highly experienced stuttering clinicians who were asked to evaluate the likelihood that each child was stuttering. Table II summarizes the statistical results of the diagnostic scores of these five experienced clinicians, of the examining clinicians who prepared the original reports, and the of decision support system for each of the 25 reports. The five experienced clinicians are represented by the letters a, b, c, d, and e. The examining clinicians who evaluated each child are designated by ec, and the decision support system by dss.

Table II
Clinical Results

Expert	Bias	Dispersion
a	-0.11	0.76
b	0.17	0.79
c	-0.17	0.71
d	0.50	0.88
e	0.14	0.86
ec	-0.42	0.85
dss	-0.10	0.80

Based on the data in Table II, the performance of the decision support system is indistinguishable from that of experienced clinicians. Its dispersion is higher than some clinicians but lower than others. Its bias is very small. In contrast, the examining clinicians' diagnostic scores do stand out from the panel of clinicians, as is indicated by the large negative bias score. This seems reasonable, because the clinicians who conducted these evaluations had access to data that the other clinicians lacked: they examined the actual children. Furthermore, they likely reviewed video tapes of their evaluation and discussed difficult cases with other clinicians before arriving at a diagnosis and writing their clinical reports. Therefore their diagnostic opinions may be the most valid. The decision support system's diagnoses were designed to fall between those of the examining clinicians and the mean of the panel of experienced clinicians. The low bias and dispersion of the decision support system indicates its robustness.

When experienced clinicians have an opportunity to confer and discuss their diagnostic opinions, differences in opinion that they may have usually decrease. At the time of their discussions, panel members were shown both the mean and range of diagnostic scores for each case. The intended purpose of these discussions was to clarify information about a case, not to develop a consensus, but the range of diagnostic scores was smaller after the discussions. This reduction in range may be due to panel members considering additional diagnostic factors during their discussions than were considered when their initial diagnoses were given. It may also reflect, in part, some panel members' decisions to modify their initial diagnostic scores to more closely approximate the panel's mean score. One purpose of Second Opinion is to allow an inexperienced clinician to "discuss" findings obtained from a diagnostic evaluation with a panel of experts and to learn the diagnostic opinions of the panel of experts. Such "discussions" are intended to increase the reliability of inexperienced clinicians' diagnoses.

The output of Second Opinion is robust. Three people used it with the same twenty five clinical reports described above. One was the chief scientist for our decision support systems, who has had over twenty five years of experience evaluating children with fluency problems. The second was a doctoral candidate in Communication Sciences and Disorders at Syracuse University who had acquired substantial previous experience with the decision support system. The third was a master's degree student at the University of Arizona who had little experience evaluating young stutterers and no previous experience with any decision support system. Their diagnostic opinions evidence little difference. The case about which there was the most disagreement, Couzens, led to further review of that clinical report and the identification of ambiguities that may have caused this discrepancy. Use of Childhood Stuttering: A Second Opinion, appears to promote diagnostic opinions that are indistinguishable from those of a panel of five experienced clinicians regardless of a clinician's training and background. One significant finding is that the diagnosis of three people with widely differing backgrounds evidenced less variability when using the decision support system than did those of five experienced clinicians rating the same clinical reports without the decision support system.

As a penultimate test, the system was given to 30 students in an Expert Systems class at the University of Arizona. It is now undergoing its ultimate test as it is used by customers. Most of the concerns of both students and customers involve the installation procedure. No one yet has pointed out errors or suggested improvements in its knowledge base.

This decision support system is meant to assist, not replace, inexperienced speech-language clinicians. One thing a human does that a computer system does not do is track a child's behavior over time, looking for improvement or a worsening of signs and symptoms. In the future we intend to build a version of this system for teaching that will include explanations and pertinent references for each question asked. Such systems would appear to be useful in training clinical skills as well as in supporting the skills of inexperienced clinicians.

VII. Conclusions

It is difficult to detect errors in a decision support system. Therefore, much effort was devoted during the knowledge extraction process to insure that errors did not creep into the knowledge base. Because all such knowledge came from and was captured by fallible humans, it is unreasonable to expect a knowledge base to be free of errors. Therefore, several tools and procedures were developed to help detect errors in a decision support system's knowledge base. Each was designed to work with existing tools, and each added additional complexity to testing procedures, and, we think, credibility to the decision support systems' performance. Finally we compared the output of the system to the evaluations of human experts. When using *Childhood Stuttering: A Second Opinion*, three clinicians with widely differing backgrounds in stuttering produced diagnostic opinions that evidence little variability and were indistinguishable from those of a panel of five experienced clinicians.

In the software industry, the manufacturing process itself is subsumed in significance by the development process. While the former requires physical quality control such as uncontaminated storage media and so on, rigorous validation and verification at the development stage is what ensures the quality of the final product. In the case discussed in this paper, the need for the domain experts' knowledge to control development while simultaneously ensuring the technical soundness of the knowledge base was of paramount concern. To this end, several innovations, reported above, had to be made. While the generalizability of such innovations need further examination, the point we seek to make is simply this: within the overall need for rigorous verification and validation, the specific method(s) to be used are often dictated by the specifics of the product and the circumstances in which it is built.

Acknowledgments

This work was supported by the National Institute of Child Health and Human Development Grant R44 HD26209 and by Bahill Intelligent Computer Systems. The contents of this paper are solely the responsibility of the authors and do not necessarily represent the official views of the National Institute of Child Health and Human Development.

REFERENCES

L. Adelman, 1992. *Evaluating Decision Support and Expert Systems.* New York: John Wiley & Sons.

Jafar, M. and Bahill, A. T. 1991. Interactive verification and validation with Validator. Chapter 4 in Bahill, A. T. ed. *Verifying and Validating Personal Computer-Based Expert Systems.* Englewood Cliffs: Prentice Hall.

Jafar, M. and Bahill, A. T. 1993. Interactive verification of knowledge-based systems. *IEEE Expert* 8(1): 25-32.

Kang, Y. and Bahill, A. T., 1990. A tool for detecting expert system errors, *AI Expert* 5(2): 46-51.