Pergamon

S0952-1976(95)00018-8

Engng Applic. Artif. Intell. Vol. 9, No. 3, pp. 275–284, 1996

0952-1976/96 $15.00 + 0.00

Contributed Paper

Performance Evaluation of a Large Diagnostic Expert System Using a Heuristic Test Case Generator

AVELINO J. GONZALEZ
University of Central Florida, U.S.A.

UMA G. GUPTA
East Carolina University, U.S.A.

RICHARD B. CHIANESE
Westinghouse Electric Corp., U.S.A.

(Received May 1995)

Validating the performance of a knowledge-based system is a critical step in its commercialization process. Without exception, buyers of systems intended for serious purposes require a certain level of guarantees about system performance. This is particularly true for diagnostic systems. Yet, many problems exist in the validation process, especially as it applies to large knowledge-based systems. One of the biggest challenges facing the developer when validating the system's performance is knowing how much testing is sufficient to show that the system is valid. Exhaustive testing of the system is almost always impractical due to the many possible test cases that can be generated, many of which are not useful. It would thus be highly desirable to have a means of defining a representative *set of test cases that, if executed correctly by the system, would provide a high confidence in the system's validity. This paper describes the experiences of the development team in validating the performance of a large commercial diagnostic knowledge-based system. The description covers the procedure employed to carry out this task, as well as the heuristic technique used for generating the representative set of test cases.*

Keywords: Knowledge-based systems, validation and verification (V&V), heuristics, software testing.

1. INTRODUCTION

In the last decade or so, a large number of successful prototypes of knowledge-based systems have been developed for a wide variety of applications,[1] in domains ranging from medicine and aerospace to finance and travel. Although knowledge-based systems have generally proved to be a promising and vibrant technology, there are relatively few systems that are operational. Partly to blame for this situation is the difficulty still being faced by system developers in providing high confidence in the correct performance of the system being developed. An inaccurate and unreliable knowledge-based system can destroy its credibility, and clearly defeats the purpose of creating an "intelligent system".[2] Although in the last decade or so, research in issues related to testing, validation and verification of knowledge-based systems has progressed greatly,[3–11] many critical issues that contribute to the development of robust and reliable systems still remain to be addressed. Furthermore, it is interesting to note that while several "how to" papers have been written on validating and verifying (V&V) knowledge-based systems, there are only a handful of papers that describe actual experiences related to the testing and performance validation of real-world, operational systems.[12–14] One significant bottleneck frequently encountered in the transition from a prototype to an operational system is the lack of a rigorous and unified framework for testing knowledge-based systems.[15–17]

Correspondence should be sent to: Professor Avelino J. Gonzalez, University of Central Florida, Department of Electrical and Computer Engineering, Orlando, FL 32816-2450, U.S.A.

With the intention of addressing the above deficiencies, this paper:

(1) describes the overall testing process followed in the validation of a large, real-time diagnostic knowledge-based system which is presently in successful commercial operation;
(2) describes a technique that generates a *representative* set of test cases, a non-exhaustive set of cases intended to provide a high probability of system validity at a reasonable cost in effort when all are correctly executed by the system; and
(3) discusses some lessons learned during the transition of the subject knowledge-based system to commercial operation.

2. VALIDATION AND VERIFICATION OF KNOWLEDGE-BASED SYSTEMS

Validation and verification (V&V) of conventional software is a well-researched area.[18-20] Issues such as software quality assurance, assertion checkers, proof of correctness, debugging aids, symbolic execution, test data generation, formal verification techniques, reliability analysis, and testing have been extensively addressed in the literature.[18,21-24]

Although the lessons from testing conventional software are both rich and valuable, they cannot be directly transferred to knowledge-based systems, due to the significant differences between conventional software and KBS. Unlike conventional software, knowledge-based systems solve problems that cannot be easily solved through algorithmic techniques. Moreover, unlike conventional software in which data and procedures are intermingled, knowledge and control are two distinct entities in knowledge-based systems. The reasoning mechanism in a KBS attempts to mimic the knowledge and expertise of the human expert, while conventional programming captures the nature of highly structured and routine decision-making. Furthermore, the ever-present and often complex interface with a domain expert can make evaluation of the system difficult. Additionally, as indicated by Shaw and Woodward, "care must be taken to ensure that it is the system that is being validated, rather than the expert from whom the knowledge was elicited".[2]

Chandrasekaran[25] lists four main purposes for testing and evaluating a knowledge-based system. Two of these are: (1) to demonstrate the performance of the system to members outside the development group, such as users, prospective clients, etc.; and (2) to identify any inherent weakness in the system. The procedure described here focuses on these two issues.

The terms *validation and verification* have been defined in Adrion *et al.*'s classic paper.[21] The definitions of these terms are briefly summarized below.

Validation: Validation determines the correctness of the final software product with respect to the user's needs and requirements.[21] It is the final quality-control step of knowledge-based system development, and ensures that the output of the system is correct (however that is defined), and that the developed system is what the users want and need. Validation can be considered to be largely an *extrinsic* measurement (as defined by Hayes–Roth)[26] of the performance of the system.

Verification: "... is the demonstration of the consistency, completeness and correctness of the software". It is ensuring that the system is built right.[27] Verification is more of an *intrinsic* measurement (also as defined by Hayes–Roth).[26]

Although the field of validation and verification of knowledge-based systems has progressed since the early 1980s, when V&V of knowledge-based systems was not being given its due consideration, a survey of the history and growth of knowledge-based systems shows that in the early eighties scientists and researchers in the AI community were focusing their efforts on issues such as knowledge-based system design, knowledge acquisition, knowledge representation, and development methodologies. It was only in the mid- to late-1980s, as more prototypes were being introduced into their operational environments, that the issues of the validation, verification, and testing of knowledge-based systems gained the attention of AI researchers.

3. DESCRIPTION OF GenAID

Although a large number of knowledge-based system prototypes has been developed in the last decade or so, very few systems have become operational. For a variety of reasons such as high implementation cost, lack of management interest, lack of confidence in the performance of the system, or inability to provide performance guarantees, a number of prototypes failed to progress into fully-fledged operational systems. One of the earliest knowledge-based systems developed as a commercial product for external users was GenAID.[1,28] An acronym for Generator Artificial Intelligence Diagnostics, GenAID was developed in the 1980s by Westinghouse Electric Corp., a manufacturer of large electric power-generating equipment, in order to help its customers decrease the downtime of their turbine generators through the early detection of potentially serious abnormal operating conditions.

Historical evidence indicates that many serious and costly turbine generator failures initially start out as relatively minor problems that go undetected for a comparatively long period of time, partially because of the inability of operators to recognize symptomatic problems in the early stages. This often led to a deterioration of the situation, and eventually resulted in a

serious problem. It has been recognized that for many failures, corrective action at an early stage would have resulted in significant reduction of damage, lowering the consequent downtime from several months to a few days.

Large power-generation equipment is typically well instrumented, with sensors that are continuously monitored by a data-acquisition system. Periodic inspection of the data by an expert would facilitate the detection of incipient failures, which in turn would lead to timely corrective action. In most cases, however, the lack of diagnostic expertise prohibited the timely detection of pending disasters. GenAID has helped to overcome this serious bottleneck by providing much-needed diagnostic expertise to the plant operators.

GenAID is a forward-chaining, rule-based system, comprising approximately 6000 rules. Originally implemented in LISP on a VAX system, it was later reimplemented in C for speed and portability. The overall system is composed of seven different modules, each representing specific parts of the generator system. These are: (1) the water-cooled stator windings; (2) the gas-cooled stator windings; (3) the hydrogen auxiliary system; (4) the seal oil system; (5) the excitation system; (6) the stator coil cooling water system; and (7) the bearing lubrication system. Depending on the type and design of the generator being monitored, however, not all seven modules need be implemented for any one generator. Some of these modules are mutually exclusive, while others are necessary only for some types of generators. The sizes of the modules range from a minimum of 200 rules to more than 1000 rules. Due to inherent limitations in the sensors that monitor the generators, there is some uncertainty regarding the accuracy of the input values, which in turn affects the reliability of GenAID's diagnosis. Hence, GenAID's diagnoses are ranked according to a numeric value which represents the likelihood of the problem being present (e.g. certainty factors). Nearly a dozen experts were involved in the knowledge-acquisition process, due to the various sub-systems within the generator, each expert representing expertise in different subsystems.

In over eight years of field operation on several units, GenAID has been successful in diagnosing a number of problems which might otherwise have gone undetected and resulted in serious incidents.[29] As a result, Westinghouse has proceeded to extend this concept to other pieces of equipment that it manufactures, such as the steam turbine (TurbinAID).

GenAID's clients consist of electric utilities and other purchasers of power-generation equipment who are heavily staffed by experienced technical personnel. Therefore, they do not merely look for general assurances about system performance, but rather, for formal, rigorous validation procedures. This was the impetus for the methodology introduced for the validation of GenAID, which is described below.

4. METHODOLOGY EMPLOYED FOR THE VALIDATION OF GenAID

Conventional software testing has traditionally involved the execution of a set of benchmarks (e.g. test cases) whose results are precisely known: the calculations of a mathematical program can often be verified through hand calculations, or the behavior of physical phenomena can be observed in an actual system and compared to that of the computer model of the same phenomenon. If the simulation produces the correct results for a given set of inputs, then it has correctly solved the test case. If this continues for the entire set of test cases, then the software system can be considered valid. The benchmark approach has worked, and continues to work, satisfactorily for traditional software.

A modified version of the benchmark approach was employed in the validation testing of GenAID. The modification consists in the fact that diagnostic test cases which use certainty factors cannot be precisely determined to be right or wrong. Various experts may have varying opinions, which may be contradictory, as shown during the validation testing of MYCIN[30] and various other systems. Thus, a more flexible means of determining the correctness of GenAID's solution to a particular test case was required.

The validation strategy for GenAID consists of the following steps:

(1) Generate a representative set of test cases to provide adequate coverage for each of the modules. Execute the test cases on the system, and analyze the results with the help of a domain expert and the knowledge engineer.
(2) Carry out design reviews of the GenAID system. This is to be done by presenting the set of test cases, as well as GenAID's answers to them, to a panel of experts. This is to be followed for each of the modules independently. The design review panel will determine whether GenAID's answer to each test case is *correct* or *incorrect*. All incorrectly solved test cases will require that a modification be made to GenAID so that it is correctly solved.
(3) Field test the system in its operational environment to detect any other problems.

A detailed discussion of the above steps follows below.

4.1. Generation and execution of test cases

The generation of test cases is a critical process in system testing. The number and quality of the test cases has a direct and significant impact on the reliability and robustness of the system. Exhaustive testing, although generally desirable, is impractical, since a very large number of test cases must be executed and evaluated, even in small systems. Hence, the goal of the GenAID

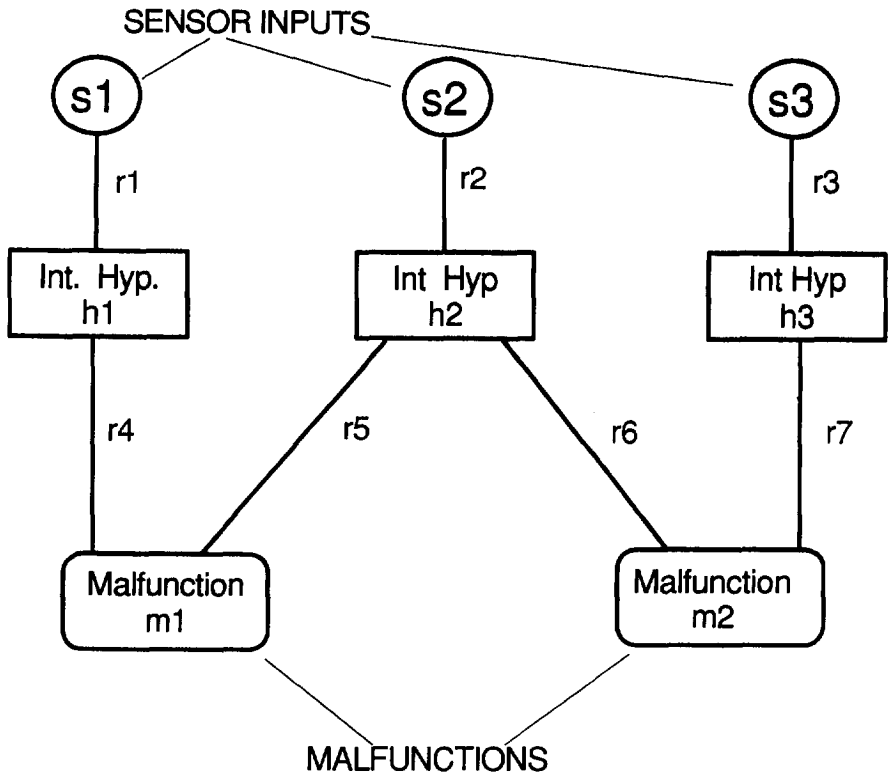

Fig. 1. Self-contained, reasonably-sized knowledge sub-module.

developers was to generate a large enough set of test cases, that would not only be highly resource-efficient but also ensure the success of the system.

The following set of test objectives was established, based on intuition and on engineering judgment:

(1) The test cases should ensure GenAID's ability to identify accurately, *at all times* malfunctions that are considered *critical*. Critical malfunctions are those that could result in either bodily injury to plant personnel or serious damage to the generator. In other words, without any exception, GenAID should accurately identify critical malfunctions and suggest expert remedies. Hence, a 100% accuracy record was required in critical malfunctions, failing which the usefulness of the product would be negated and its credibility compromised.

(2) The test cases should correctly identify "*most of the time*" (i.e. 80–90% of the time depending on the nature of the system) malfunctions that occur frequently, but are not considered critical. Malfunctions such as sensor failures and routine maintenance items would fall in this category. These malfunctions are such that, at their worst level of severity, they would not represent a significant unit outage, or potential injury to personnel.

(3) The test cases should correctly identify, most of the time (i.e. more than 50% of the time) all other malfunctions that are neither common nor critical in nature.

Different priorities were assigned to the above three objectives, with the first one having a *high* priority, the second one *medium* priority, and the last, *low* priority.

4.2. Division of GenAID into sub-modules

In order to implement the objectives listed in Section 4.1 above in a manageable fashion, the tactic of divide-and-conquer was employed. Each module in GenAID was divided into sub-modules, and test cases were generated for each of these.

Two kinds of sub-modules are defined here: *self-contained/reasonably-sized* sub-modules, and *overlapping* sub-modules. These are described below.

Self-contained/reasonably-sized sub-modules: Groups of related knowledge elements (i.e. rules, intermediate hypotheses, malfunctions, sensors) which could be grouped together without any overlap with other such groups were called *self-contained sub-modules*. They were kept to a reasonable size (not more than two or three malfunctions) for manageability, thus the adjective *reasonably-sized*. Figure 1 depicts a reasonably-sized/self-contained sub-module of a knowledge-base module. Segregation of the knowledge in this fashion helped generate the set of test cases for each sub-module in an easily managed fashion.

Overlapping sub-modules: Self-contained sub-modules that grew too large were divided into smaller

sub-modules that shared some knowledge element. These were called *overlapping sub-modules*. Figure 2 shows one example where it is not possible to create reasonably-sized self-contained sub-modules.

Overlapping sub-modules were treated slightly differently, since some test cases in other overlapping sub-modules would require values of a sensor which were already repeated in the tests for the neighboring sub-module. This repetition was not expected, however, for sensors in self-contained sub-modules since they did not relate to anything outside their own sub-module. This made the generation of test cases for self-contained sub-modules somewhat easier. A formal representation of the above concept is described as follows.

The entire knowledge base, KB, is defined as a set composed of the various modules described above, called M_i. The KB for each generator instance includes those modules that are applicable to that particular generator, such that

$$\mathrm{KB} = \{M_1, M_2, M_3, \ldots, M_i\}.$$

Similarly, each module M_i is a set composed of sub-modules (self-contained as well as overlapping), SM, such that

$$M_i = \{SM_{i,1}, SM_{i,2}, SM_{i,3}, \ldots, SM_{i,j}\}.$$

In turn, each sub-module is a set of basic knowledge elements defined as malfunctions (m), rules (r), intermediate hypotheses (h), and input sensors (s), which are also grouped together as subsets. Thus, this can be described as

$$SM_{i,j} = \{\{m_1, m_2, m_3, \ldots, m_k\}\{r_1, r_2, r_3, r_4, \ldots, r_l\} \{h_1, h_2, h_3, \ldots, h_n\}\{s_1, s_2, s_3, s_4, \ldots, s_p\}\}$$

$$SM_{i,j+1} = \{\{m_{k+1}, m_{k+2}, \ldots, m_{k+a}\}\{r_{l+1}, r_{l+2}, \ldots, r_{l+b}\} \{h_{n+1}, h_{n+2}, \ldots, h_{n+c}\}\{s_{p+1}, s_{p+2}, \ldots, s_{p+d}\}\}$$

$$SM_{i,j+2} = \{\{m_{k+a+1}, m_{k+a+2}, \ldots \text{etc.}\} \ldots\},$$

where i corresponds to the module of which all the sub-modules are subsets, and j is the label of each particular sub-module.

By definition, the intersection of the set of rules for any one sub-module with that of another sub-module should be the empty set. This is likewise true for malfunctions, hypotheses and sensors, and applies to self-contained as well as overlapping sub-modules. Therefore, no malfunction, rule, hypothesis or sensor appears in more than one module or sub-module. The union of the malfunction, rule, hypothesis and sensor subsets for all the sub-modules in a module makes up the knowledge in an entire module.

The malfunctions are now further defined as a tuple, where the first element is the malfunction's own label, and the second term is a set of the rules that act to set its value. This can be described as

$$\langle m_x\{r_y, \ldots, r_z\}\rangle,$$

where x stands for the number assigned to a particular malfunction, and y and z stand for the names of any

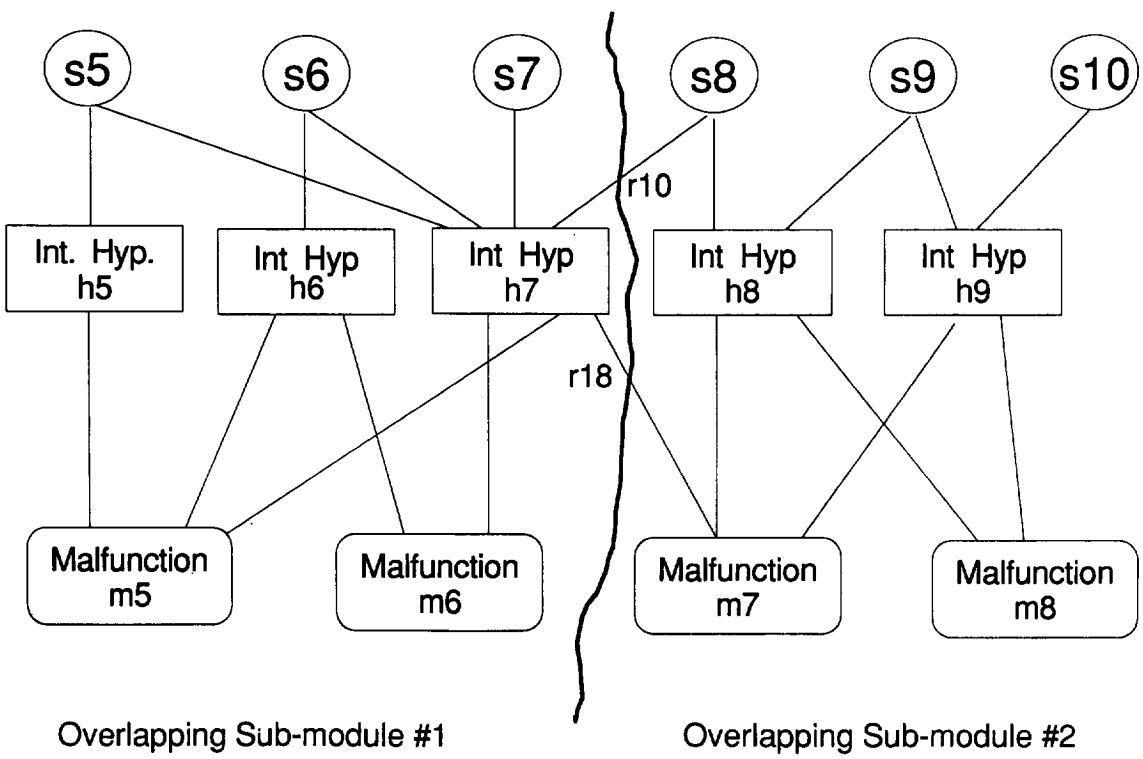

Fig. 2. Overlapping sub-modules.

rules that are capable of determining the value of m_x. It is important to note that the rules in this set, called the *set-by rules*, do not have to be in the same sub-module, or even the same module, as the malfunction. The sensors can similarly be described as tuples whose first term is the label of the sensor, and the second a set of the rules that use that sensor in their premises (called the *used-by rules*), which also do not have to belong to the same sub-module as the sensor. Intermediate hypotheses can be described as triples, where the second term is the set of all used-by rules, while the third term represents the set of set-by rules.

Assuming that the sub-module of Fig. 1 is labeled $SM_{1,1}$ (a subset of module M_1), it could be thus represented as follows

$$SM_{1,1} = \{\{m_1, m_2\}\{h_1, h_2, h_3\}\{s_1, s_2, s_3\} \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}\},$$

where m_1 can be represented as $\langle m_1 \{r_4, r_5\}\rangle$, h_2 can be represented as $\langle h_2 \{r_5, r_6\} \{r_2\}\rangle$, and so on.

Using this representation, then, a self-contained sub-module can be defined to be one in which the union of all the set-by and used-by rules belonging to its malfunctions, hypotheses and sensors is exactly equal to its set of rules. A sub-module that does not meet this criterion is considered an overlapping sub-module. Note that in Fig. 2, rule r_{10} belongs to sub-module 2, yet is found in the set of set-by rules for hypothesis h_7. Likewise, rule r_{18} is part of sub-module 1, yet appears in the set of set-by rules of conclusion (malfunction) m_7.

4.3 Test case generation heuristics

Once the modules have been divided into smaller units, the next step is to generate the representative test cases for each of these sub-modules using heuristics.

Two specific heuristics were used. The first one, called the *classification heuristic*, was used to classify each malfunction being tested according to its priority level as described in Section 4.1: *high priority*, *medium priority*, and *low priority*. The classification heuristic is used to dictate how thoroughly to test each of the malfunctions within the sub-module of interest.

The second heuristic is actually a set of three heuristics which guides the generation of test cases for each sub-module according to the priority level of each malfunction therein. These are as follows:

High-priority malfunctions were very thoroughly (although not exhaustively) tested. This was intuitively appropriate, since customers could not tolerate any errors in diagnosing these malfunctions. This heuristic divided the test cases into two groups of tests:

(1) Each path was individually tested four times with the following input data for the applicable sensor:
   - — Once just above its upper limit (AUL).
   - — Once just below its lower limit (BLL).
   - — Once in the normal portion of the range (NR).
   - — Once in the alarm portion of the range (AL).

   Any other sensor which affects the malfunction being tested through a different path is set to normal values, since that path is not being tested.

   For example, if a water temperature sensor had a maximum realistic value of 100°C (steam), a low realistic value of 0°C (ice), a normal range of between 30° and 60°, and an alarm point of 65°, then the four test points would be: 110°, −5°, 40° and 65°C. This would serve to validate the influence of each sensor reading individually on each malfunction. It should be noted that the first two test points should not result in the identification of the malfunction, since they will most likely indicate faulty sensors.

(2) The powerset of all non-individual combinations of paths to each high-priority malfunction were tested. Table 1 depicts a series of tests that includes the complete set of such combinations for one malfunction (of possibly several) in a knowledge sub-module. Each combination was tested with a combination of sensor values, where each sensor took on each of its four possible values, as described above. This resulted in nine additional tests for the example of Fig. 1, since some combinations had already been tested with the individual paths.

Table 1. Complete set of combinations of paths to be tested for malfunction m_1 of Fig. 1 if it were a high priority malfunction

Test no.	S_1	S_2	S_3	Paths tested
1	AUL	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
2	BLL	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
3	NR	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
4	AL	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
5	NR	AUL	n/a	$s_2/r_2/h_2/r_5/m_1$
6	NR	BLL	n/a	$s_2/r_2/h_2/r_5/m_1$
7	NR	AL	n/a	$s_2/r_2/h_2/r_5/m_1$
8	AUL	AUL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
9	AUL	BLL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
10	AUL	AL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
11	BLL	BLL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
12	BLL	AUL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
13	BLL	AL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
14	AL	AUL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
15	AL	BLL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$
16	AL	AL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$

The number of tests would have been significantly greater if there had been more than two paths affecting the particular high-priority malfunction being tested. For example, if there had been three paths affecting m_1

(say, paths p_1, p_2, and p_3), the powerset of combinations would include

$$\{p_1, p_2, p_3, p_1/p_2, p_2/p_3, p_1/p_3, p_1/p_2/p_3\},$$

where each combination of two paths would be similar in nature to tests 8–16 in Table 1.

For **medium-priority malfunctions**, the complete set of combinations of paths for each malfunction used in high-priority malfunctions was also used, but the sensor inputs used were only two: one at normal conditions and one at alarm. If the assumption is made that malfunction m_1 in Fig. 1 is a medium-priority malfunction, the set of test cases is shown in Table 2.

The number of test cases would have also been somewhat higher if the malfunction being tested had more than two paths affecting it. However, since the number of sensor values used in the tests is not as comprehensive as for high-priority malfunctions, the number of tests would not be nearly as large as for a high-priority malfunction.

Finally, for **low-priority malfunctions**, only each individual path was tested, with one sensor reading in the normal range and one in the alarm range. See Table 3.

Since no combinations of paths are tested for the low-priority malfunctions, the number of tests would be insensitive to multiple (>2) paths affecting the malfunction.

Repetitive groups of the knowledge base, that is, identical knowledge used for symmetrical parts of a generator, were basically ignored. This was justified because the basic concept could be tested by only testing one such group. For example, in a generator containing 96 coils, each of which could be independently monitored and diagnosed, there would be 96 identical groups of knowledge. Successful testing of one of these could safely be extended to include validation of all 96.

This procedure provided a "formula", based on intuition and expressed through heuristics, that can be used to determine an appropriate set of test cases to be generated for the validation of each module. As a result of applying these heuristics, the smaller modules required somewhere in the neighborhood of 50–75 cases. This was due to the fact that there were very few critical cases in these. The larger modules contained more critical cases, but also contained a certain number of repetitive knowledge groups which were not explicitly tested beyond the first group. The number of test cases for these larger modules was in the neighborhood of 100.

Table 2. Complete set of combinations of paths to be tested for malfunction m_1 of Fig. 1 if it were a medium-priority malfunction

Test no.	S_1	S_2	S_3	Paths tested
1	NR	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
2	NR	NR	n/a	$s_2/r_2/h_2/r_5/m_1$
3	AL	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
4	NR	AL	n/a	$s_2/r_2/h_2/r_5/m_1$
5	AL	AL	n/a	$s_2/r_2/h_2/r_5/m_1 - s_1/r_1/h_1/r_4/m_1$

Table 3. Complete set of combinations of paths to be tested for malfunction m_1 of Fig. 1 if it were a low-priority malfunction

Test no.	S_1	S_2	S_3	Paths tested
1	AL	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
2	NR	NR	n/a	$s_1/r_1/h_1/r_4/m_1$
3	NR	NR	n/a	$s_2/r_2/h_2/r_5/m_1$
4	NR	AL	n/a	$s_2/r_2/h_2/r_5/m_1$

The heuristics used are flexible, and can easily be modified to make the test more or less stringent, depending on the resources available. Nevertheless, the procedure as described above proved to be satisfactory for the GenAID validation.

It should be mentioned, however, that this methodology only detected *errors of commission*; that is, those errors induced through the execution of the knowledge in the knowledge base. *Errors of omission*, those errors which result from the absence of knowledge in the knowledge base, cannot be discovered using the procedure outlined above. Errors of omission are manifested when a combination of inputs fails to identify a malfunction that should have been detected. Discovering errors of omission required the involvement of experts to detect any gaps in the knowledge. This is discussed in Section 4.4 below.

4.4. Test case design review

A set of test cases was generated using the above heuristics, and was then executed on the system. The performance of GenAID was then validated using a formal *design review* of each module. Design reviews are quality-assurance procedures, common in the engineering of large or critical systems, in which a panel of knowledgeable individuals carefully scrutinizes and formally assesses a design before the product is commercially produced.

The design review panel for each module consisted of four experts in the technical domain pertinent to that module: two were part of the original knowledge engineering team, and two were outside experts who had had no role in the development of GenAID (generally brought in from field service divisions of Westinghouse). In addition, design review panel members included a representative from the Quality Assurance Department, as well as the knowledge engineer.

The test requirement that GenAID must accurately execute *all* test cases is significant, as it eliminated the need to conduct probabilistic analysis to determine the accuracy of the system.

Quite frequently, system developers tend to evaluate the validity of the system based on the percentage of test cases found to be correct. However, this alone is insufficient to ensure the reliability of the system. A truly *representative* set of test cases lies at the heart of

performance validation, and such a set should take into account not only the number of test cases, but also their nature. Anything less than this would, in the opinion of the authors, constitute inadequate coverage of the system.

At the end of each design review, members of the review panel were asked to identify any malfunctions, or combination of sensor readings leading to a malfunction that may have been overlooked by the test cases. This was the means of identifying errors of omission. Any new test case suggested by the experts was added to the test case set for that module, and executed immediately, and the results were reviewed by the panel. An average of two to three test cases were added in each design review as a result of this process.

Although initially there were some concerns that such an "open forum" approach might lead to situations where there was irreconcilable disagreement between experts, surprisingly, the experts seemed to be in agreement on most of the significant issues. One possible reason for this could be that the domain of turbine generator diagnostics is quite mature, and the experts working on the project all had significant experience.

4.5 Final validation of the system through field testing

Clearly, validation of each module independently does not guarantee that the entire system has been validated. Moreover, some aspects of the system, such as the characteristics of the sensing instruments, and their effects on system behavior, cannot be adequately validated through test cases alone. Thus, the GenAID developers felt strongly that field testing was essential to ensure the proper performance of GenAID. The independence of its component modules greatly facilitated incremental field testing of GenAID. This is, in fact, one of the more significant advantages of partitioning the knowledge base into system-related modules. Upon completion of the design review of a module, it was placed on-line in its operational environment, and its performance was continuously monitored and communicated to the development team. Any diagnosis generated, whether correct or otherwise, was not communicated to the customer until carefully confirmed by the development team. Errors detected during field testing were quickly traced, the knowledge base revised, and the revision documented before the revised module was incorporated into the main (operational) system.

After a "probationary" period of field testing, the new module was considered operational, and its results were made directly (and immediately) available to the customers. This probationary period served mainly to ensure that uncalibrated instruments, or their sporadic and/or unduly erratic behavior, would not cause false alarms. The length of the probationary period depended on the number and types of operating conditions "seen" by GenAID. For example, it was important that GenAID be subjected to a period of high generator load, and of low generator load, an off-line incident and to a start-up experience. The probationary period typically lasted 2–3 months. The approval of the performance of each module by the development team was required before the module was integrated with the main system, and the results of its diagnosis were directly forwarded to the customer.

5. EVALUATION OF VALIDATION METHODOLOGY USED

This section summarizes the results obtained, and discusses some of the contrasting experiences of the developemnt team when validating a sister knowledge-based system, TurbinAID.

Although approximately 60% of the test cases for each module were considered unacceptable by the design review panel, the errors which caused the deficient label for most of these were rather minor in nature (i.e. the certainty factor assigned to the malfunction being tested by that test case was inappropriate). These errors were quite easy to correct, and in some cases, the knowledge base was revised overnight, in time for the next morning's panel meeting.

Approximately 10% of the test cases in a typical module, however, were deemed unacceptable for more significant reasons, such as incorrect diagnosis, or incorrect sensor reading threshold. While some of these were comparatively easy to correct, they generally represented a more complex situation than those discussed above.

The knowledge base was modified so that all unsatisfactory test cases executed correctly prior to probationary field testing. Most modules passed their probationary on-line operation with only a few (2–3) totally incorrect diagnoses.

Although GenAID has produced some incorrect diagnoses in its operational environment, these have been very few in number (estimated to be less than 20 in over 70 unit/years of operation). More notably, none of these errors have been critical. This was highly consistent with the validation objectives declared at the outset of the process. For the above reasons, it can be safely said that the validation of GenAID was successful.

A sister system to GenAID, called TurbinAID, steam Turbine Artificial Intelligence Diagnostics, trailed the GenAID development process by approximately 18 months, and presented an interesting contrast to GenAID. Unlike generators, turbines are much more diverse in design and application. Each design has unique features which make it susceptible to different malfunctions. As a result, the experts tend to remember specific issues which may have been observed in one design, but do not apply to other designs. Thus, the same set of symptoms can easily elicit different diag-

noses from different experts, depending upon their personal experience.

Furthermore, the normal operation of turbines is characterized by highly dynamic sensor readings. Normal operating temperatures and pressures can vary widely, depending upon load and steam cycle operating characteristics. Even a perfectly operating turbine takes approximately 30 min to reach steady-state conditions. The operation of the turbine auxiliary system can also have a significant impact on sensor readings.

Moreover, not only are the sensor readings in a turbine much less predictable than for generators, but to further complicate matters, the number and variety of turbine sensors is quite large. Most power plants do not have the level of instrument coverage necessary to diagnose all possible conditions adequately, so expert diagnosticians tend to make assumptions that are often incorrect. Only during the past decade have computerized data-acquisition systems been applied to accumulate and record sensor readings from all parts of the plant. Thus, turbine diagnosticians have not traditionally had good data to use in the diagnosis of malfunctions. Therefore, the concept of diagnosing turbine conditions at the nascent stages of a developing malfunction from on-line sensor readings is beyond the practical experience of many turbine experts.

What is interesting about the above is that, in spite of the significant differences in the type of domain knowledge between the GenAID and the TurbinAID systems, the validation procedure pioneered by the GenAID system was successfully applied to various modules of the TurbinAID system. The validated modules have proceeded to go into commercial operation, and represent a competent representation of the domain knowledge. This lends some credence to the universal applicability of the validation procedure described in this paper.

6. CONCLUSIONS AND SUMMARY

This paper has presented a description of experiences in validating a commercially successful knowledge-based system. These are embodied in the procedure used to carry out this task. Nevertheless, the success achieved in validating the GenAID system has introduced a general technique for carrying out the same process on other knowledge-based systems. This process is based on the following elements:

- The use of heuristics to develop a set of test cases that is representative of the established validation criteria. The set of test cases is also meaningful to ensure a representative (but not exhaustive!) coverage of the knowledge base.
- The use of a design review format to analyze the results of the test cases and ensure that all test cases are correctly solved by the knowledge-based system.

There is no theoretical basis for these heuristics. Their development was based on intuition and on the experience of the GenAID developers. The fact that these were sufficient in the successful validation of GenAID and TurbinAID only points to their suitability in the validation process of these two systems. However, the general concepts behind them can potentially be extended to the validation of other knowledge-based systems. Certainly, the details of the heuristics (how many sensor readings are to be used, etc.) can be modified to fit some other criteria or requirements placed on the target knowledge-based system.

In general, the approach presented here describes a systematic procedure to provide feedback for the system developers. It was used successfully in the validation of GenAID and TurbinAID, and it appears that it can be effective for other types of knowledge-based systems.

REFERENCES

1. Feigenbaum E., McCorduck P. and Nii H. P. *The Rise of the Expert Company*. Times, New York (1988).
2. Shaw M. and Woodward J. Validation in a knowledge support system. Construing and consistency with multiple experts". *Int. J. Man Machine Studies* **29**, No. 3, pp. 329–350 (1988).
3. Suen C. Y., Grogono P. D. and Shingal R. Verifying, validating and measuring the performance of expert systems. *Expert Systems with Applications* **1**, 93–102 (1990).
4. Gupta U. G. (ed.), *Validating and Verifying Knowledge-based Systems*. IEEE Computer Society Press, Las Alamedas, CA (1992).
5. Meseguer P. and Plaza E. An overview of the VALID project. *Personal Computing and Intelligent Systems* (Edited by E. H. Vogt) Information Processing, Vol. III. Elsevier, Amsterdam (1992).
6. Zlatareva N. A framework for knowledge-based system, verification, validation and refinement: The VVR system. *Proc. 5th Florida Artificial Intelligence Research Symp.*, Ft. Lauderdale, pp. 10–14 (1992).
7. Preece A. D., Suen C. Y. and Yu C. L. Performance assessment of a character recognition expert system. *Proc. 2nd Expert Systems Application World Conference*, pp. 103–109 (1990).
8. O'Leary T. J., Goul M., Moffitt K. E. and Radwan A. E. Validating expert systems. *IEEE Expert* **5**, No. 3, 51–58 (1990).
9. Cohen P. R. and Howe A. D. Toward AI research methodology: three case studies in evaluation. *IEEE Trans. on Systems, Man and Cybernetics* **19**, No: 3, 634–646 (1989).
10. Brown D. E. and Pomykalski J. Reliability estimation during prototyping of knowledge-based systems, Institute for Parallel Computation, University of Virginia, Charlottesville, pp. 1–23 (1991).
11. Benbasat I. and Dhaliwal J. A framework for the validation knowledge acquisition. *Int. J. Man–Machine Studies* **1** , No. 2, 215–233 (1989).
12. Ludvigsen P. J. and Dupart R. R. Formal evaluation of the expert system DEMOTEX. *J. Computing Civil Engng* **2**, No. 4, 398–412 (1988).
13. Hershauer A. K., Owens H. and Philippakis A. A field observation study of an expert system prototype development. *Information and Management*, Vol. 17, pp. 107–116 (1989).
14. O'Neil M. and Glowinski A. Evaluating and validating very large knowledge-based systems. *Medical Information* **15**, No. 3, pp. 51–58 (1990).
15. Bundy A. How to improve the reliability of knowledge-based systems. *Proc. Knowledge-based Systems '87,* the 7th Annual Technical Conference of the British Computer Society Specialist Group on Knowledge-based Systems, pp. 3–17. Cambridge University (1987).
16. Culbert C., Riley G. and Savely R. T. Verification issues of rule-based expert systems. *Proc. Third Conf. on Artificial Intelligence*

for Space Applications, National Aeronautics and Space Administration Washington, D.C. (1987).
17. Green C. J. Verification and validation of expert systems. *WESTEX-1987*, pp. 38–43 IEEE Computer Society Press (1987).
18. Howden W. E. Errors in data processing programs and the refinement of current program test methodologies. Final Report, National Bureau of Standards, contract NB79BAC0069 (1981).
19. Wallace D. R. and Fujii R. U. Software verification and validation: an overview. *IEEE Software*, May, pp. 10–17 (1989).
20. Boehm B. W. Verifying and validating software requirements and design specifications. *IEEE Software*, January (1984).
21. Adrion W. R., Branstad M. A. and Cherniavsky J. C. Validation, verification and testing of computer software. *Computing Systems* **14**, No. 2, 159–192 (1982).
22. DeMillo R. A. and Offutt A. J. Constraint-based automatic test data generation. Technical Report, GIT-SERC-87/17, Software Engineering Research Center, Georgia Institute of Technology (1987).
23. Hetzel W. C. An experimental analysis of program verification methods. Ph.D. dissertation, University of North Carolina, Chapel Hill (1976).
24. Miller M. Verification and validation of decision support expert systems: process. *American Chemical Society Expert System Applications in Chemistry Symp.*, Los Angeles, CA, pp. 126 (1988).
25. Chandrasekaran B. Generic tasks in knowledge-based reasoning: high level building blocks for expert system design. *IEEE Expert*, pp. 23–30 Fall (1986).
26. Hayes-Roth F. Towards benchmarks for knowledge systems and their implications for data engineering. *IEEE Trans. Knowledge and Data Engng* **1**, No. 1, March (1989).
27. O'Keefe R. M., Balci O. and Smith E. P. Validating expert systems performance, *IEEE Expert* **2**, No. 4, Winter, pp. 81–90 (1987).
28. Gonzalez A. J., Osborne R. L., Kemper C. and Lowenfeld S. On-line diagnosis of turbine-generators using artificial intelligence. *IEEE Trans. Energy Conversion***EC-1**, No. 2, pp. 68–74 June (1986).
29. Thompson T. and Coffman M. Texas utilities electric experience with on-line generator monitoring and diagnostics. *American Power Conf.*, Chicago, IL (1988).
30. Buchanan B. G. and Shortliffe E. H. Uncertainty and evidential support. In *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project* (Edited by B. G. Buchanan and E. H. Shortliffe), pp. 571–588. Addison-Wesley, Reading, MA (1985).

AUTHORS' BIOGRAPHY

Dr Avelino J. Gonzalez received his Bachelor's and Master's degrees in electrical engineering from the University of Miami, in 1973 and 1974 respectively. He obtained his Ph.D. from the University of Pittsburgh in 1979, also in Electrical Engineering. He spent nearly 12 years with Westinghouse Electric Corp. in Pittsburgh, PA and Orlando, FL, working in computer applications to electrical power engineering, and developing the Westinghouse generator diagnostic expert system GenAID, for which he received the Westinghouse Award for Excellence in Engineering. In 1986, Dr Gonzalez joined the Computer Engineering Department (which later became the Electrical and Computer Engineering Department) at the University of Central Florida (UCF) in Orlando, where he has focused his research and teaching in artificial intelligence and knowledge-based systems. Dr Gonzalez is a registered Professional Engineer in the State of Florida.

Dr Uma G. Gupta is an Associate Professor in the department of Decision Sciences at East Carolina University. She holds a Ph.D. in Industrial Engineering, and an MBA from the University of Central Florida, as well as a Master's Degree in mathematics from India. Dr Gupta's research interests include testing case-based systems and expert systems, and managerial and organizatinal issues relating to management information systems. She is the author of *Management Information Systems: A Managerial Perspective* (West Publishing), and has edited a book on validating expert systems. Dr Gupta has more than 40 refereed journal articles and converence proceedings to her credit.

Dr Richard B. Chianese is a senior engineer at the Power Generation Business Unit of Westinghouse Electric Corporation in Orlando, FL. He has 25 years of experience in the design, analysis and quality assurance of power-generation equipment. From July 1986 to December 1992, he was a member of the Diagnostics and Monitoring Department, where he had responsibility for developing and verifying the TurbinAID expert system described in this paper.