Research Feature

Managing Conflict in System Diagnosis

As modern systems become more complex, it becomes more likely that system maintainers will be presented with conflicting information when testing them. Maintainers need advanced techniques to make an accurate diagnosis of the problem, especially in the face of large numbers of conflicts. This article describes two approaches to resolving conflicting test results: modified Dempster-Shafer statistical inference and certainty factors.

John W. Sheppard
ARINC

William R. Simpson
Institute for Defense Analyses

The complexity of modern systems has led to new demands on system diagnostics. As systems grow in complexity, the need for reliable testing and diagnosis grows accordingly. The design of complex systems has been facilitated by advanced computer-aided design/computer-aided engineering (CAD/CAE) tools. Unfortunately, test engineering tools have not kept pace with design tools, and test engineers are having difficulty developing reliable procedures to satisfy the test requirements of modern systems.

The testing of complex systems is rarely perfect. In software, it is almost impossible to track system state or all the ways values might be affected. Hardware systems are frequently subject to noise and other random events, which makes it difficult to interpret test results and thus lowers the confidence in what the tests indicate. Even with digital testing, which eliminates some noise problems, developers must still contend with the effects of state. Finally, modern systems depend heavily on both hardware and software, and the interactions between hardware and software further compound the problem of managing test errors.

When testing a complex system, what is the proper response to unexpected and conflicting test results? Should the results be scrapped and the tests rerun? Should the system be replaced or redesigned? Should the test procedures be redeveloped? Determining the best answers to these questions is not easy. In fact, each of these options might be more drastic than necessary for handling conflict in a meaningful way.

When test results conflict, the potential source of the conflict must be analyzed to determine the most likely conclusion. To date, test systems have done little more than identify when a conflict exists. Since the early 1970s, artificial intelligence researchers have attempted to "handle" uncertain and conflicting test information. But this handling has been limited to assigning probabilities or confidence values to the conclusions, providing a ranked list of alternative actions.[1,2] When test results are uncertain but consistent, this is about the best we can do.

In this article, we discuss two approaches to system diagnosis that apply several tests and interpret the results based on an underlying model of the system being tested. These tests are used to determine if the system is functioning properly and, if not, to explain the faulty performance. When test information conflicts, ignoring or improperly handling the conflict will degrade diagnostic accuracy. By examining potential sources of conflict and the way conflict might become manifest in a reasoning system, we developed an approach to extend diagnosis to handle the conflict and draw more reliable conclusions.

TRADITIONAL DIAGNOSTICS

Frequently, test engineers define a system-level diagnostic process that is independent of the design and manufacturing process. The first step, for example, is to develop built-in test (BIT) or built-in self test (BIST), which make the initial detection and localization of faults. When used with other tests, these tests—which are embedded in the system itself—may localize faults to a level sufficient to take action.

Subsequent steps apply a battery of automatic and manual tests to the system (or subsystem). Eventually, these tests might identify the subunit suspected of containing the fault. The subunit is then tested to find the faulty unit. Once a unit or subunit is separated from the system, maintainers frequently use specialized equipment (usually from the unit manufacturer) to test it.

Despite improvements in BIT, BIST, and automatic testing, manufacturers typically have not provided maintainers with comprehensive diagnostic procedures. Instead, they rely on part screenings and spe-

0018-9162/98/$10.00 © 1998 IEEE

Despite improvements in BIT, BIST, and automatic testing, manufacturers typically have not provided maintainers with comprehensive diagnostic procedures.

cial test approaches, which are inadequate because they emphasize proper system function rather than the isolation of a fault when the system does not function properly. (One exception is the screening of complex incoming parts such as chips, which is much more comprehensive. This is due to both improved manufacturing reliability and the complexity of testing a large microchip.)

This approach to system testing is an artifact of a manufacturing process that tests only pieces of systems. It is clearly insufficient to explain anomalous behavior at the system level, as it fails to account for the complex interactions among system components. At this level, we are left with a few euphemisms that describe heuristic approaches, such as "tickle testing" (when we snug all fittings and clean all contacts) or "shotgun maintenance" (when we guess where the fault resides and take action until the system anomalies disappear).

INTEGRATED DIAGNOSTICS

In developing an alternative to this approach, we focused on ideas developed in *integrated diagnostics* programs. Integrated diagnostics emphasizes the application of structured approaches to system testing and diagnosis. Such programs have three objectives:

- to maximize the reuse of design and test data, information, knowledge, and software;
- to integrate support equipment and manual testing, so that complete coverage of diagnostic requirements is covered; and
- to integrate the available diagnostic information, so that resources are minimized and performance optimized.

Our research focuses on applying a single uniform method for representing diagnostic information: One model type represents the system at all levels of detail. Using this model, test engineers can determine BIT requirements, define test programs for automatic test equipment, and guide the manual troubleshooting process.

The model we use captures test information flow: It models the information provided by a set of tests with respect to a set of desired conclusions. During troubleshooting, the information gathered from performing the series of tests is combined to make a diagnosis. Defining the relationships between tests and conclusions results in an *information flow model*. The models are hierarchical, in that a conclusion in one model can be used to invoke a lower level model. The rules for handling each model and submodel are the same, regardless of their position in the hierarchy.

We begin by developing a set of information flow models for the system to be tested. We develop models for on-board diagnosis (thus determining the requirements for BIT) and for each subsequent level of testing. The conclusions we draw at one level determine the appropriate model to use at the next level.

Once developed, we analyze the models to evaluate the system's testability. We verify specification compliance and perform design trade-offs to improve testability. Thus the modeling process begins in the early stages of system development. As the system progresses through the life cycle, the models are revised to reflect changes and refinements in the design. For diagnosis, the models define available tests and inferences that can be drawn by obtaining test outcomes. Hence, the same models used to evaluate testability of the system can be used for troubleshooting.

INFORMATION FLOW MODEL

An information flow model[3] represents the problem to be solved via the flow of diagnostic information. Tests provide information, and diagnostic inference combines information from multiple tests using several logical and statistical inference techniques. The structure of the information flow model facilitates our ability to compute testability measures and derive diagnostic strategies.

An information flow model has two basic elements: *tests* and *conclusions*. Tests include any source of information that can be used to determine the health of a system. Conclusions typically represent faults, including hardware fault modes, failure of functionality, specific nonhardware failures (such as bus timing), and specific multiple failures. A conclusion may also indicate the absence of a failure indication (no fault).

Information obtained during testing might be a consequence of observing system operation or a response to a test stimulus. Thus we include observable symptoms of failure in the information flow model as tests. Including these symptoms allows us to analyze situations involving information sources in addition to formally defined tests.

When developing a fault isolation strategy, test engineers should consider the type, amount, and quality of test information. We initially assumed equal quality among test results. In other words, we assumed that every test outcome actually reflects the state of the unit being tested. In practice, we relax this assumption and allow a measure of confidence to be associated with each test.

If all possible inferences drawable from performing each test are known, then the information gained by performing each test can be calculated. The set of test inferences allows us to draw conclusions about a sub-

set of components. At any point in the test sequence, the information flow model can be used to compute the set of remaining failure candidates.

In designing our model, we developed a precise algorithm to look at the information content of the tests. Specifically, we determine *test information gain* by computing the reduction in size of the failure candidate set for each test. The algorithm selects tests such that the number of tests required to isolate a fault is minimized over the set of potential failure candidates; it does this by maximizing information gain at each step in testing.

DEMPSTER-SHAFER INFERENCE

The algorithms our approach applies for drawing inferences from actual test information are a modification of Dempster-Shafer statistical inference,[4,5] which is derived from Bayesian inference theory. It has four steps:

1. We compute for every conclusion the two extremes of a credibility interval: *support* and *plausibility*. The probability that a given conclusion is true lies between its support and plausibility values. A test outcome *supports* a conclusion (and thereby its associated fault) when the outcome indicates the detection of the fault associated with that conclusion. A test outcome *denies* a conclusion if it eliminates the conclusion from consideration. Denial is the complement of plausibility. To compute a conclusion's support and plausibility values, we begin by assigning a confidence value to the test outcome. We use these confidence values to compute support and denial values. In particular, we uniformly distribute the confidence value over all conclusions supported and apply the full weight of the confidence value to all conclusions denied.
2. Using this result as a basis, we compute support and plausibility measures incrementally. We determine how much the new evidence conflicts with previously accumulated evidence (initially assuming no disagreement). Then we revise the support for each conclusion using a variant of Dempster's Rule of Combinations. Dempster's rule computes normalized mutual support and denial for each conclusion by combining the current accumulation of support and denial and the support and denial received from the most recently evaluated test. Figure 1 shows how the rule works. To determine plausibility, we keep a running average of the denial obtained thus far and subtract from one: plausibility equals 1 minus denial.
3. We modified the Dempster-Shafer process by defining a special conclusion, which we call the *unanticipated result*.[6] The unanticipated result compensates for disappearing uncertainty, even in the face of conflict—a known deficiency in the Dempster-Shafer process. Support for the unanticipated result (representing conflict) is computed whenever evidence denies the current hypothesis. For this to occur, the evidence must deny all of the conclusions in the current hypothesis set, because the hypothesis corresponds to the logical OR of the members of the set. If the current hypothesis set is denied, the amount of conflict is then computed on the basis of the number of tests executed so far. When no conflict exists, support for the unanticipated result decays with each test performed.
4. To determine the final support measure, we normalized the previously calculated support measure by the sum of all conclusion support values plus the support for the unanticipated result.

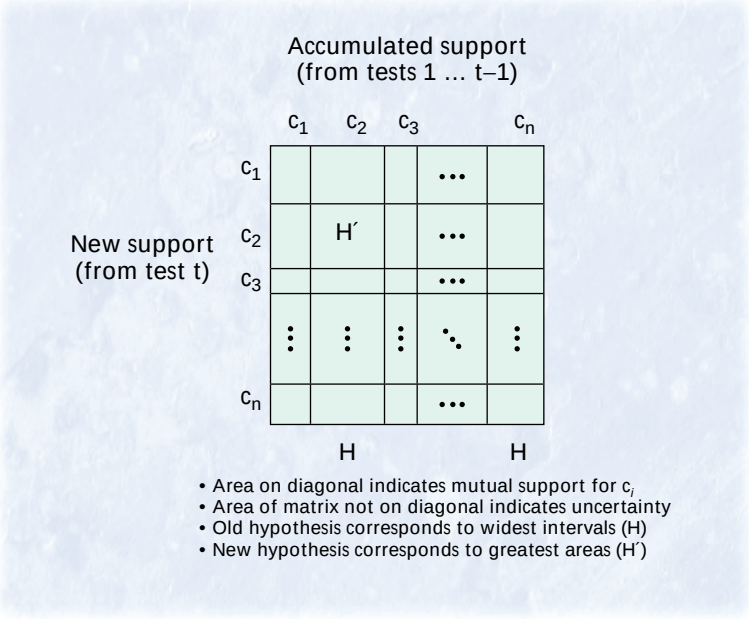

Figure 1. Dempster's Rule of Combinations.

The primary computational burden of this procedure lies in determining the normalization constant of Dempster's rule. This normalizer requires a summation over all pairwise combinations of support values. It has a complexity of $O(n^2)$, where n is the number of conclusions. The calculations for combining support and denial and for computing conflict are relatively simple, being of complexity $O(1)$, and the final calculation for normalizing support is $O(n)$. Thus, the overall computational complexity of this process is $O(n^2)$ in each step.

CERTAINTY FACTORS

Because the support value depends so strongly on previously normalized data, the Dempster-Shafer cal-

Integrated diagnostics emphasizes the application of structured approaches to system testing and diagnosis.

culations exhibit a *temporal-recency effect*. In other words, more recent events have a greater impact on the evidential calculation than more distant events. As a result, if the same set of tests is analyzed with the same outcomes and the same confidences but in different orders, the resulting Dempster-Shafer statistics might be different.

Because of this undesirable property, we explored alternative approaches to reasoning under uncertainty. We wanted to be able to base our inferences on the information flow model, assign confidences to test outcomes, and perform consistent inference, independent of temporal ordering.

To guide us, we identified the reasonable characteristics for any uncertainty-based inference system. These characteristics include

- the ability to track the levels of support and denial for each conclusion in the model;
- the ability to convert these support and denial measures in a reasonable, intuitive way, in order to estimate the probability that the conclusion is true;
- the assurance that test results applied in any order would yield the same result; and
- the ability to evaluate levels of conflict such that all measures associated with conflict have the same properties as any other conclusion in the model.

From these characteristics, we derived a simplified approach to reasoning with uncertain test data and discovered that we had rederived a relatively old method called *certainty factors*.

Certainty factors were first used by Edward Shortliffe in his Mycin diagnostic system, developed in the early 1970s. With roots in probability theory, certainty factors provide an intuitive approach to reasoning under uncertainty in rule-based systems.[7] Certainty factors satisfied all of our requirements except for handling conflict, and so we developed an approach to satisfy this requirement.[8]

CERTAINTY FACTORS ADAPTED TO DEMPSTER-SHAFER

Recall that test outcomes either support or deny conclusions in the conclusion space. Our application of certainty factors to system diagnosis differs from Dempster-Shafer because it assigns the full confidence value to all conclusions either supported or denied, rather than apportioning confidence to the supported conclusions. Obviously, support is applied to a conclusion only if the test outcome actually supported that conclusion, and denial is applied only if the test outcome actually denied the conclusion.

Updating support and denial over time is also different from the Dempster-Shafer approach and is straightforward; it is similar to combining probabilities. We update these measures by adding the current support or denial to the previously accumulated support or denial and subtracting the product of the two. We determine certainty in a conclusion by subtracting the accumulated denial from the accumulated support. We then rescale the resulting certainty value so that it lies between zero and one, so we can interpret the value like a probability.

Shortliffe's approach does not identify conflict. Our approach, on the other hand, determines conflict on the basis of levels of mutual consistency among tests in the test set. We then include the unanticipated result in the model.

Support and denial

Our approach to determining support and denial for the unanticipated result requires that we designate a *support set* and a *denial set* for each test prior to the diagnosis. The support set is defined to be the set of conclusions (or faults) that are potentially drawable or detectable, either by a test failing or the set of conclusions not being eliminated from consideration because a test passes. We determine the denial set by taking the complement of the support set, which adds no new complexity to our calculation.

Using the information flow model, for any given test we can determine the set of conclusions supported by the test when the test has a given outcome. We want to compare this support set with the support sets of other tests. In particular, for a sequence of tests, we are interested in determining the relative conflict between all pairs of tests in that sequence, under the assumption of a single conclusion being drawn at a time. Relevant and likely multiple faults can be designated as a single conclusion in the model.

For example, consider Test 1 and Test 2, which have only pass and fail outcomes. These two tests might conflict in any of four possible situations: when both tests pass, when both tests fail, when Test 1 passes and Test 2 fails, and when Test 1 fails and Test 2 passes.

Relative amount of conflict

Suppose both tests fail. If we consider the intersection of the tests' support sets given their failure, we claim that if the intersection is the empty set, these two outcomes are inherently conflicting; that is, they support completely different sets of conclusions and deny each other's sets of conclusions, as illustrated in Figure 2. Given this, we can determine the relative amount of conflict by dividing the size of the intersection (the amount of mutual support) by the size of the union (the potential amount of mutual support) and subtracting from one.

Similarly, we can determine the relative amount of conflict *denial* associated with a pair of test outcomes. If the intersection of the support sets is not empty, then there exists a set of conclusions that is mutually supported by these two test outcomes. This area of mutual support indicates that the test outcomes are inherently nonconflicting, thus indicating we can deny the presence of conflict in the diagnostic process, as shown in Figure 2. Therefore, we can compute that the relative denial of conflict between two test outcomes is the same as the relative conflict, except we do not subtract from one. Individual values for the support or denial of the unanticipated result depend on the confidence in the test outcomes and can be computed by treating the confidence values as weighting factors.

We use these comparisons between the support sets for the various outcome combinations to determine the support and denial of an unanticipated result. We update support and denial for an unanticipated result at each step in the sequence.

As tests are evaluated, we accumulate support or denial for the unanticipated result similar to combining support and denial for any other conclusion, except that a single test outcome can cause several new "events" to be added. Thus, we have to combine conflict at each step by considering all past steps in the test process. This is the most computationally expensive part of the certainty factor approach: It requires $O(T^2)$ time, where T is the number of tests performed. The complexity of computing relative conflict is $O(m^2)$ for each of the four alternatives, where m is the number of tests in the model; however, this process need be performed only once for each model.

The primary advantages to using certainty factors rather than Dempster-Shafer include reduced computational complexity and sequence independence in determining support and denial for each of the conclusions. Dempster-Shafer's primary advantage is a firmer grounding in probability theory and a larger base of practical experience demonstrating acceptable behavior.

INTERPRETING CONFLICT

Drawing conclusions from uncertain but consistent test information is relatively straightforward. However, interpreting conflicting test results can be problematic, and some basic assumptions are necessary for making a diagnosis in which conflict might arise. We make three assumptions:

- We interpret only those test results that limit the possible outcomes of a test to be either pass or fail.
- We assume that the diagnostic system focuses on identifying a single fault.
- We reject the assumption that the model is correct and the test results (or at least their confidences) are correct.

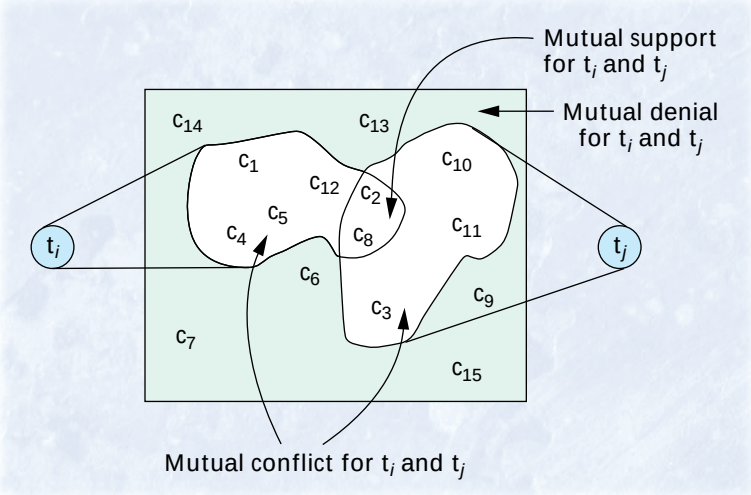

Figure 2. Determining support and denial for an unanticipated result.

Causes of conflict

We believe there are only three fundamental reasons for conflict:

- An error occurred in testing or test recording
- Multiple faults exist in the system
- The diagnostic model or underlying fault model is incomplete or inaccurate

We claim all other "reasons" can be reduced, ultimately, to one of these three reasons.

By providing a separate conclusion in the model for conflicting information (that is, the unanticipated result), we offer a powerful mechanism for identifying whether one of these situations exists. We point out, however, that independent analysis might be required to distinguish these potential sources of conflict in any particular instance. Three approaches have been used in actual diagnosis to identify causes of conflict: identifying testing errors, identifying multiple faults, and identifying modeling errors.

Identifying testing errors

Automatic testing takes much of the uncertainty associated with test performance out of testing because the same test is applied every time, and a machine performs and evaluates the results of the test. Unfortunately, automatic testing, while eliminating many sources of error, introduces many more sources.

First, it uses software, which must be written and tested. Until recently, much of the test software had been written from scratch for every new system test, leading to high development costs and a high likelihood of repeated mistakes.

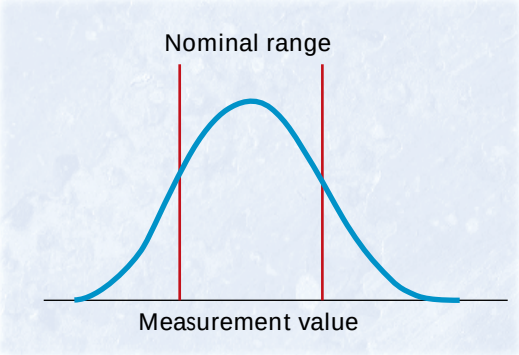

Figure 3. Nominal range for a value.

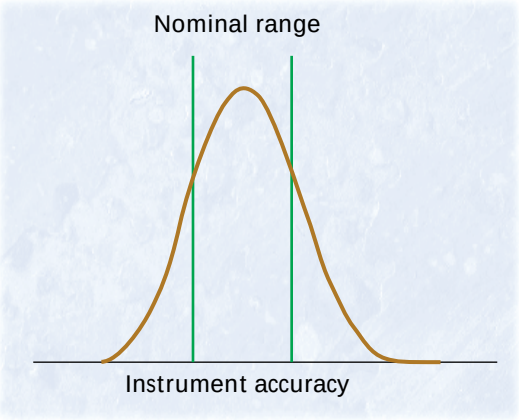

Figure 4. Nominal range for an instrument.

Second, the instrumentation used to apply test stimuli (inputs) and interpret responses (outputs) has physical limitations that can introduce errors. For example, suppose we are measuring a value in a system that might be subject to noise. The expected nominal value might fit a normal distribution (assuming Gaussian noise), and the actual nominal value should appear within a specified range, as Figure 3 shows. The instrument measuring the output might not be able to interpret the output with sufficient accuracy or precision, which also introduces errors. The nominal range for the instrument, therefore, is modeled in a similar fashion, as Figure 4 shows. The problem arises when we overlay these nominal ranges, as Figure 5 shows.

If the actual value falls on the inside of the nominal range but we declare that the test fails, we have introduced a *false alarm*. On the other hand, if the actual value falls outside the nominal range and we declare that the test passed, we have introduced a *false assurance*. In statistics, these errors are referred to as Type I and Type II errors; correspondence of a false alarm to Type I or Type II depends on whether our hypothesis is that the system is nominal or faulty.

Sensitivity to Type I and Type II errors depends on the variance of the noise associated with a measurement and the variance on the measurement device itself. When errors exist in comparing the measured value to the test limits, the potential for conflict in diagnosis increases. Inconsistent test results lead to conflict or to an incorrect diagnosis. The modified Dempster-Shafer approach provides a means for identifying the conflict and for gathering supporting or denying evidence through additional testing. For example, in an aircraft radar diagnostic system developed applying this approach,[9] accumulated BIT was regarded as a single application of an associated test procedure. Because accumulated BIT relies on repeat evaluation of the associated test procedure, confidence in that test is dependent on the number of repeat failures that occur. When the total number of failures recorded nears the threshold, we might see an increase in support for the unanticipated result.

Identifying multiple faults

The next step in managing conflict involves identifying the possibility of multiple faults in the system being tested. Multiple fault diagnosis is highly complex,[6,10,11] but many of today's diagnostic systems claim to diagnose multiple faults. Actually, it is possible to *estimate* most likely multiple faults fairly efficiently in many cases. Nevertheless, the ability to correctly and efficiently isolate the multiple fault *every* time one occurs is impossible mathematically. Support from the reasoning process to guide the search for multiple faults can be extremely valuable.

We use the conflict conclusion, or unanticipated result, as an indication that a multiple fault might be present. If diagnosis leads to a single fault conclusion with no conflict, the probability is high that the single fault identified is correct (ignoring the problems of masked root cause faults and false failure indications). Therefore, if testing yields inconsistent results, one possible cause is the presence of multiple faults.

Conflict occurs when two or more results contradict each other. Such conflict is manifest in Dempster-Shafer when a test result denies the current hypothesis and in certainty factors when the support sets for two tests are disjoint. Typically, when examining the ranked list of faults returned by either Dempster-Shafer or certainty factors, the technician treats them as ambiguous. When conflict occurs, this flags the technician that more than one of the faults in the list might be present. Generally, the technician considers the faults in order of probability. This approach can still be used. In addition, with the indication of conflict, the diagnostic system can apply a separate process among the top-ranked faults to determine if their combined symptoms match the observed symptoms. If so, this is a strong indication that the multiple fault might be present.

The radar system[9] we tested provided several examples of possible multiple faults. In one case, the problem was the presence of two antenna faults: Two

independent BIT codes were obtained directly, indicating two separate failure modes of the antenna. The diagnostic system noted the conflict and called for a separate *initiated* BIT (IBIT) to be run; that is, a set of built-in tests invoked by a technician while the airplane was on the ground. This test identified the antenna and both faults were equally supported in the fault candidate list. At the flight line, the multiple fault was benign: The individual faults were contained within the same replaceable unit. Without recognizing the multiple fault, however, personnel repairing the removed antenna would have had a problem finding the actual cause of the antenna failing.

Identifying modeling errors

Diagnostic modeling and fault modeling are extremely difficult and error-prone. It is also extremely difficult to identify errors in the diagnostic model or deficiencies in the fault model. So far, the best approaches for model verification have been based on an analysis of the model's logic characteristics and the insertion of faults in the diagnostic strategies.

Fault insertion is generally the most effective approach but it can be time-consuming and even damaging. Using fault simulation together with fault insertion eliminates the problem of potentially damaging the system. However, it adds to the verification problem because now the simulation model must also be verified.

One of the advantages of modeling systems with the information flow model is that a model's logical characteristics can be extremely useful in verifying it. Identifying ambiguity groups, redundant and excess tests, logical circularities, masked and masking faults, and potential false failure indications can all indicate potential problems with a model.

Even with the best verification tools and techniques, errors invariably remain (especially in complex models). During actual testing, it is frequently difficult to identify when model errors arise. However, including the unanticipated result in the set of possible conclusions allows a flag to be raised whenever a model error might have been encountered, because support for the unanticipated result would be nonzero. As with test error and multiple fault diagnosis, if the model is in error, at some point a test result will likely be contrary to what is expected—in which case a conflict occurs and the unanticipated result gains support.

For the radar system,[9] we found model errors because two types of models were available for comparison—a system-level model and multiple replaceable unit-level models. (Although the two types of models were not derived independently, they offered alternative representations of the system.) In one case, when test results were processed through the model representing the transmitter, no conflict was encountered and a large ambiguity group was identified (the wiring, the transmitter itself, and the power supply). When the same test results were run through the system-level model, the same ambiguity group was concluded as the most likely fault, but considerable conflict was encountered during testing. No conflict was encountered in the replaceable unit model.

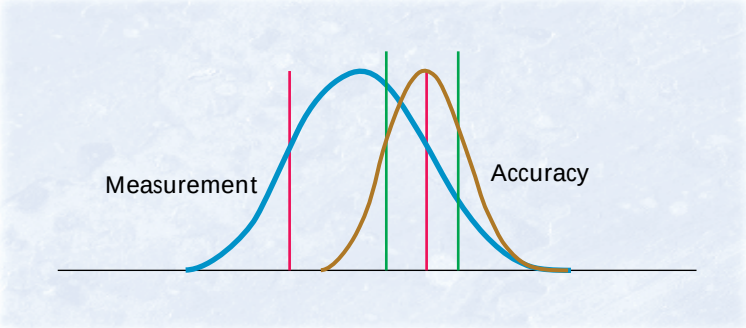

Figure 5. Uncertain pass/fail criteria.

We addressed this discrepancy by looking at the diagnostic log files for the two runs. Only one test was considered for the replaceable unit-level model, and that test pointed directly at the fault in question. The failing test determined if the transmitter was transmitting by examining the least significant bit in a data word sent in a loop-back test. At the system level, seven tests were considered (including the one from the replaceable unit level). Four of these tests did not appear in the replaceable-unit level model. Six tests, including these four, depended on the common test in the system-level model and all passed where the common test failed. Because each of these tests depended on the common test, when the common test failed each of these additional tests should have failed as well. Because they did not, they were in conflict with previous experience.

This discrepancy led us to take a closer look at the tests in question. One of the conflicting tests that appeared in both models served as a "collector" of test information in the BIT. In other words, all test information was funneled through that test such that if any other test failed, that test was expected to fail as well. But this test had another odd characteristic. Whenever certain of these other tests failed, this test was disabled and not run. The diagnostic system assumed that a test that was not run would have passed (a bad assumption) and this led to the conflict. This problem was solved in the model by linking the failed test to the tests being skipped such that the skipped tests were no longer considered, thus correcting the model.

Diagnostic modeling and diagnostic testing is complex and often leads to conflicting information. Traditional fault tree-based approaches might miss apparent conflicts and yield inferior diagnostic accuracy. Whether interpreting uncertain test

results, handling multiple faults, or contending with modeling errors, the diagnostics should support gathering information capable of identifying conflict and assessing the cause of the conflict.

Because modeling and knowledge engineering are complex and error-prone tasks, the diagnostics can be used to assist modeling by identifying when inconsistent conditions arise. Such inconsistency is identified in the diagnostics with the unanticipated result gaining support. If the certainty in the test results is correctly represented and no multiple fault exists in the system, then an analyst can assume that a problem exists in the model.

Test outcomes can result in a wide range of possible inferences, and unexpected inferences can be identified with conflict. By examining the tests evaluated and the conclusions drawn, the potential cause of the conflict can be localized and the possible problems in the model identified. On the other hand, if the problem is not within the model, the unanticipated result can guide analysis of the test process to determine unreliable tests or to identify multiple faults.

To date there has been little discussion of the positive role of conflict in the testing and diagnosis of complex systems. Conflict cannot be avoided, only ignored, and ignoring conflict results in a loss of diagnostic information. Conflict can provide valuable information about the tests, the diagnostics, and the system being tested. The approaches herein can be used to capture and quantify the conflict in the testing of complex systems. Then the resulting information can be used to identify errors or improve the diagnosis itself. Thus these approaches provide a method for using the conflict to benefit the testing process, leading in turn to more robust overall system diagnostics. ❖

Acknowledgments

This article describes the results of work that has been performed over several years, and we have received input and guidance from several people to improve the techniques and the article itself. We thank Brian Kelley, John Agre, Tim McDermott, Jerry Graham, Don Gartner, and Steve Hutti for their comments as the algorithms were developed and the system fielded. We also thank Elizabeth Reed, the four anonymous reviewers, Jim Aylor, and the editors of the IEEE Computer Society, whose comments helped to significantly improve this article.

References

1. J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo, Calif., 1988.
2. Y. Peng and J. Reggia, *Abductive Inference Models for Diagnostic Problem Solving*, Springer-Verlag, New York, 1990.
3. J.W. Sheppard and W.R. Simpson, *System Test and Diagnosis*, Kluwer Academic, Norwell, Mass., 1994.
4. A.P. Dempster, "A Generalization of Bayesian Inference," *J. Royal Statistical Society*, Series B, 1968, pp. 205-247.
5. G. Shafer, *A Mathematical Theory of Evidence*, Princeton Univ. Press, Princeton, N.J., 1976.
6. J.W. Sheppard and W.R. Simpson, "Multiple Failure Diagnosis," *Proc. Autotestcon*, IEEE Press, New York, 1994, pp. 381-389.
7. E.H. Shortliffe, *Computer Based Medical Consultations: Mycin*, Elsevier, New York, 1976.
8. J.W. Sheppard, "Maintaining Diagnostic Truth with Information Flow Models," *Proc. Autotestcon*, IEEE Press, New York, 1996, pp. 447-454.
9. D. Gartner and J. Sheppard, "An Experiment in Encapsulation in System Diagnosis," *Proc. Autotestcon*, IEEE Press, New York, 1996, pp. 468-472.
10. J. deKleer, "Diagnosing Multiple Faults," *Artificial Intelligence*, Vol. 28, 1987, pp. 163-196.
11. M. Shakeri et al., "Sequential Test Strategies for Multiple Fault Isolation," *Proc. Autotestcon*, IEEE Press, New York, 1995, pp. 512-527.

John W. Sheppard is a staff principal analyst at ARINC, where he is responsible for internal and contract research and development in intelligent testing and diagnosis. He is the author of more than 100 publications, including the first book on system diagnostics. Sheppard received a BS in computer science from Southern Methodist University and an MS and a PhD in computer science from Johns Hopkins University. He is a member of the IEEE Computer Society.

William R. Simpson is a professional staff member at the Institute for Defense Analyses, where he is involved in defining software architectures and the commercial standards associated with automatic testing systems. He has written more than 180 publications and is the coauthor, with John Sheppard, of System Test and Diagnosis *(Kluwer Academic Publishers, 1994). Simpson received a BS in aerospace engineering from Virginia Polytechnic Institute and State University, an MS and a PhD in aerospace engineering from Ohio State University, and an MSA in engineering administration from George Washington University. He also graduated from the US Naval Test Pilot School.*

Contact Sheppard at ARINC, 2551 Riva Rd., Annapolis, MD 21401; jsheppar@arinc.com. Contact Simpson at IDA, 1801 N. Beauregard St., Alexandria, VA 22311; rsimpson@ida.org.