

Chapter 6

Managing Conflict in System Diagnosis

John W. Sheppard
ARINC Incorporated

William R. Simpson
Institute for Defense Analyses

Keywords: Diagnostic inference models, diagnostics, system test, certainty factors, Dempster-Shafer, conflict management

Abstract: Advanced diagnostic techniques have become essential for supporting reliable testing of modern systems due to the rapid growth in complexity of these systems. Errors in the test process become more likely with increased complexity and need to be treated carefully to maximize accuracy in diagnosis. In this paper, we discuss several approaches to analyzing conflicting test results. We have concentrated on one approach based on model-based diagnosis—the information flow model—to identify conflict and improve diagnostics.

1. INTRODUCTION

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.

Testing of complex systems is rarely perfect. In software, it is almost impossible to track system state or all of the ways values might be affected. Hardware systems are frequently subject to noise and other random events making interpretation of test results difficult, thus lowering 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 handling the uncertainty and the conflict has been limited to assigning probabilities or confidence values to the conclusions to provide a ranked list of alternatives for taking appropriate action.^{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 system performance. When test information conflicts, ignoring or improperly handling this 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.

2. SYSTEM DIAGNOSIS

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) for initial detection and localization of faults. These tests, which are embedded in the system itself, when used with other tests, 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 screening and special test approaches, which are inadequate in that they emphasize ensuring the system functions properly rather than isolating faults when the system does not function properly.

This approach to system testing is an artifact of a manufacturing process that tests only pieces of systems. This approach is clearly insufficient to explain anomalous behavior at the system level, since it fails to account for the complex interactions among system components. At this level, we are left with a few euphemisms to 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).

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

- Maximum reuse of design and test data, information, knowledge, and software.
- Integration of support equipment and manual testing, to provide complete coverage of diagnostic requirements.
- Integration of available diagnostic information, to minimize required resources and optimize performance.

Our research focuses on applying a 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 defined for the system 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 the *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 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 drawn 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 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.

3. THE INFORMATION FLOW MODEL

To address several problems associated with performing system diagnosis and analyzing system testability, we introduced the concept of an information flow model (Simpson & Sheppard, 1994). This model-based approach to system test and diagnosis incorporates techniques from information fusion and model-based reasoning to guide analysis. The model represents the problem to be solved as diagnostic information flow. Tests provide information, and diagnostic inference combines information from multiple tests using information fusion and statistical inference. The structure of the information flow model then facilitates our ability to compute testability measures and derive diagnostic strategies.

An information flow model has two primitive elements: *tests* and *fault-isolation conclusions*. Tests include any source of information that can be used to

determine the health state of a system. Fault isolation conclusions include failures of functionality, specific non-hardware failures (such as bus timing), specific multiple failures, and the absence of a failure indication (*No Fault*). The information obtained may be a consequence of the system operation or a response to a test stimulus. Thus, we include observable symptoms of failure processes in the information flow model as tests. Including these symptoms allows us to analyze situations that involve information sources other than formally defined tests. Of course, the purpose of the model is to combine information obtained from these information sources (tests) to derive conclusions about the system being diagnosed.

When developing a fault isolation strategy, the type, amount, and quality of test information should be considered. For our purposes, we initially assume equal quality among test results in the sense that the *good* or *bad* indication of a test actually reflects the state of the unit under test. During actual diagnosis, we relax this assumption to allow a confidence value to be associated with a test result. If all test inferences in a system are known, the information content of each test can be calculated. If a test is performed, the set of inferences allows us to draw conclusions about a subset of components. At any point in a sequence of tests, the model can be used to compute the set of remaining failure candidates. We developed a precise algorithm to look at the information content of the tests. This algorithm selects tests such that the number of tests required to isolate a fault is minimized over the set of potential failure candidates.

4. DIAGNOSIS USING THE INFORMATION FLOW MODEL

Fault isolation can be mathematically described as a set partition problem. Let $\mathbf{C} = (c_1, c_2, \dots, c_n)$ represent the set of components. After the j^{th} test, a fault-isolation strategy partitions \mathbf{C} into two classes. $\mathbf{F}^j = (c_1^j, c_2^j, \dots, c_m^j)$ is the set of components that are still failure candidates after the j^{th} test (feasible set). $\mathbf{G}^j = \mathbf{C} - \mathbf{F}^j$ is the set of components found to be good after the j^{th} test (infeasible set). By this structure, a strategy will have isolated the failure when \mathbf{F}^j consists of a single element or an indivisible component ambiguity group.

Let \mathbf{D} represent the full set of test inference relationships between components and test points. This is formulated as a matrix representation. Let \mathbf{S}_k be a sequence of k tests, (t_1, t_2, \dots, t_k) . Let \mathbf{F}^k be the feasible failure candidate set associated with \mathbf{S}_k . We then develop an information measure, \mathbf{I}_k^j , for each remaining (unperformed) test, t_j , which is a function of the inference relationship and the remaining candidate failure

class, say, $\mathbf{I}_k^j = f(\mathbf{D}, \mathbf{F}^k)$ (Shannon, 1948; Dretske, 1982; Quinlan, 1986; Simpson & Sheppard, 1994). The test sequence \mathbf{S}_k that is derived is obtained by optimizing at each decision point. That is, the next test in the sequence is taken as the test that maximizes \mathbf{I}_k^j for the conditions imposed by each previous test outcome and is based on an unknown current outcome. The sequence ends when adequate information is derived for fault isolation. Although this algorithm uses the greedy heuristic (i.e., does local search), it is based upon a higher order representation and has been providing performance near the theoretical optimum (Simpson & Sheppard, 1994).

5. DIAGNOSIS AND CONFLICT MANAGEMENT WITH DEMPSTER-SHAFER INFERENCE

Our approach to diagnosis uses a modification of Dempster-Shafer (Dempster, 1968; Shafer, 1976) statistical inference in its inference engine (Simpson & Sheppard, 1994). To summarize, we compute values for two extremes of a credibility interval for every conclusion in the model. These extremes are called *Support*, s_{c_i} , and *Plausibility*, p_{c_i} , and for a given conclusion, c_i , $s_{c_i} \leq \Pr(c_i) \leq p_{c_i}$. To compute these measures, we begin with assigning a confidence value to a particular test outcome, cf_{t_j} . In our formulation, we uniformly distribute the support over all conclusions supported and apply the full weight of denial (the complement of plausibility) to all conclusions denied. Thus,

$$s_{c_i} = \frac{cf_{t_j}}{|\mathbf{C}_s|} \quad (1)$$

$$d_{c_i} = cf_{t_j} \quad (2)$$

where \mathbf{C}_s is the set of conclusions supported by the evidence given in t_j , and d_{c_i} is the *denial* of conclusion c_i .

From these, 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 on Dempster's Rule of Combinations (Dempster, 1968) which computes normalized mutual support and denial for each conclusion

$$u(t) = u(t-1) \frac{1 - cf_{i_j}(t)}{1 - k(t)} \quad (5)$$

$$\tilde{d}_{c_i}(t) = \tilde{d}_{c_i}(t-1) + d_{c_i}(t) \quad (6)$$

$$p_{c_i}(t) = 1 - \frac{\tilde{d}_{c_i}(t)}{t} \quad (7)$$

where,

$$\delta_{ij} = \begin{cases} 0; i = j \\ 1; i \neq j \end{cases}. \quad (8)$$

A modification to the Dempster-Shafer process includes defining the *unanticipated result* (Simpson & Sheppard, 1994). This special conclusion helps to compensate for declining uncertainty in the face of conflict. The support for an 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 hypothesis set $\mathbf{H} \in \mathbf{C}^+$ (a non-empty set of conclusions). The amount of conflict is apportioned over the number of tests executed so far, so

$$\tilde{s}_u(t) = \tilde{s}_u(t-1) + \frac{\chi(t)k(t)cf_{i_j}(t)}{t} \quad (9)$$

where χ is the number of times a conflict has occurred. When no conflict exists, support for the unanticipated result decays according to

$$\tilde{s}_u(t) = \tilde{s}_u(t-1) \frac{t-1}{t}. \quad (10)$$

Now we are ready to compute the final support measure. First note that equation 7 computes plausibility as a function of normalized denial. Since we are interested in maintaining information on raw denial and rederiving plausibility, we have nothing to be concerned about with the plausibility calculation. However, so far there has been no equivalent normalization operation for support. At each step, support is normalized as follows.

$$\tilde{s}_{c_i}(t) = \frac{\hat{s}_{c_i}(t)(1 - \tilde{s}_u(t))}{u(t) + \sum_{\forall c \in \mathbf{C}} \hat{s}_c(t)}. \quad (11)$$

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.

6. DIAGNOSIS AND CONFLICT MANAGEMENT WITH CERTAINTY FACTORS

Because of this strong dependence of the support value on previously normalized data, the Dempster-Shafer calculations exhibit a temporal-recency effect. In other words, more recent events have a greater impact on the evidential calculation than more distant events. The significance of this is that the evidential statistics are not temporally independent. As a result, if the same set of tests are analyzed with the same outcomes and the same confidences but in different orders, the resulting Dempster-Shafer statistics will be different.

Because of this undesirable property, we began to explore alternative approaches to reasoning under uncertainty in which we could base our inferences on the information flow model, assign confidences to test outcomes, and perform consistent inference independent of any temporal ordering. As a guide, we began by listing several characteristics we felt were reasonable for any uncertainty-based inference system. These characteristics included the following:

- We should be able to track levels of support and denial for each conclusion in the model.
- We should be able to convert these support and denial measures to an estimate of probability given the evidence, i.e., $\Pr(c_i|e)$ that is both reasonable and intuitive.
- We should be able to apply test results in any order and yield the same result.
- We should be able to evaluate levels of conflict in the inference process, and all measures associated with conflict should have the same properties of any other conclusion in the model.

From these “requirements,” we started to derive a simplified approach to reasoning with uncertain test data and discovered that we had re-derived a relatively old method called *certainty factors*. Certainty factors were first used by Edward Shortliffe in his *MYCIN* system, developed in the early 1970s and provided an intuitive approach to reasoning under uncertainty in rule-based systems that had several roots in probability theory (Shortliffe, 1976). As we started to work with certainty factors, we found they satisfied all of our requirements except for the handling of conflict. The following describes our implementation of certainty factors for the information flow model, including the creation of a conflict-management strategy that satisfies the above requirements.

As with Dempster-Shafer, we begin by noting that test outcomes either support or deny conclusions in our conclusion space. The first variation on Dempster-Shafer, is that we assign the full confidence value to all conclusions either supported or denied rather than apportioning confidence to the supported conclusions. Using the notation developed above for Dempster-Shafer, we have,

$$s_{c_i} = cf_{t_j} \quad (12)$$

$$d_{c_i} = cf_{t_j} \quad (13)$$

Obviously, as before, support is only applied to a conclusion if the test outcome actually supports that conclusion, and denial is only applied if the test outcome actually denies the conclusion.

Updating support and denial over time is straightforward and has similarities to combining probabilities. In particular, we can update support and denial as follows:

$$\tilde{s}_{c_i}(t) = \tilde{s}_{c_i}(t-1) + s_{c_i}(t) - \tilde{s}_{c_i}(t-1)s_{c_i}(t) \quad (14)$$

$$\tilde{d}_{c_i}(t) = \tilde{d}_{c_i}(t-1) + d_{c_i}(t) - \tilde{d}_{c_i}(t-1)d_{c_i}(t) \quad (15)$$

According to Shortliffe, the certainty in a conclusion is given by

$$cert_{c_i}(t) = \tilde{s}_{c_i}(t) - \tilde{d}_{c_i}(t). \quad (16)$$

This is not quite enough for us since $cert_{c_i} \in [-1,1]$. First we need to rescale the value such that $cert'_{c_i} \in [0,1]$. We accomplish this as follows:

$$cert'_{c_i} = \frac{1}{2}(cert_{c_i} + 1). \quad (17)$$

Then we compute the probability as

$$\Pr(c_i|e) = \frac{cert'_{c_i}}{\sum_{\forall c \in C \cup \{unt\}} cert_c}. \quad (18)$$

Note this equation includes *unt*, i.e., the unanticipated result. Recall that all test outcomes support some conclusions and deny other conclusions. Prior to doing any diagnosis, we can determine the support sets for each of the tests. Determining the denial set is done by taking the complement of the support set which adds no new information to our calculation. Further, the impact of denial is based on a single failure assumption which makes determining conflict based on denial questionable.

For any given test, we can determine the test's support set when the test passes and when the test fails. We want to compare these support sets to 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. Support and denial for conflict then consist of combining support and denial at each step in the sequence using the combination procedures described by Equations 14 and 15. All we need now is a way to determine s_u and d_u (Sheppard, 1996).

Consider two tests t_i and t_j . These two tests may conflict in any of four possible situations—when both tests pass, when both tests fail, when t_i passes and t_j fails, and when t_i fails and t_j passes. Without loss of generality, suppose both tests fail. If we consider the intersection of the tests' support sets given they fail, we claim that if the intersection is the empty set, these two outcomes are inherently conflicting, i.e., they support completely different sets of conclusions and, in fact, deny each other's sets of conclusions (Figure 2). In this scenario, we can determine the relative amount of conflict as follows:

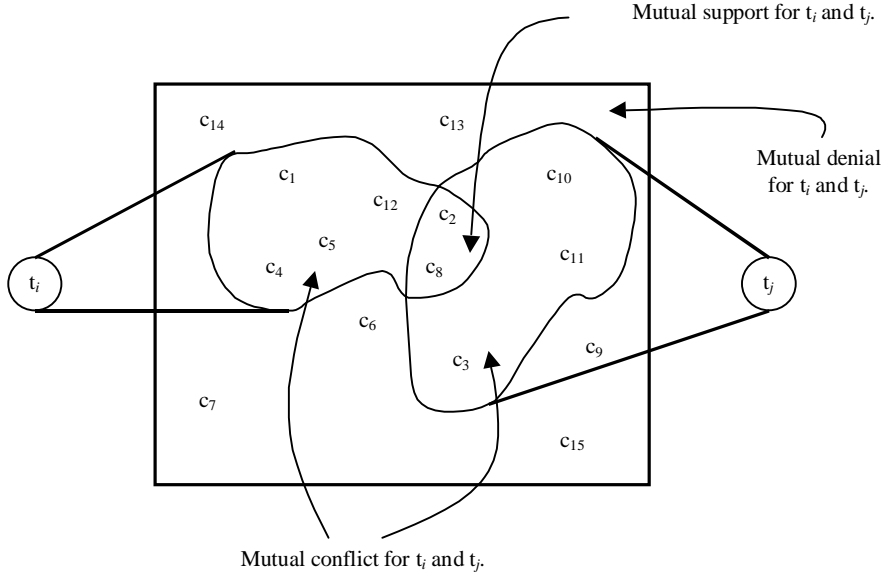


Figure 2. Determining support and denial for unanticipated result.

$$\chi(val(t_i) = \text{FAIL} \wedge val(t_j) = \text{FAIL}) = 1 - \frac{|\mathbf{C}_{t_i}^F \cap \mathbf{C}_{t_j}^F|}{|\mathbf{C}_{t_i}^F \cup \mathbf{C}_{t_j}^F|} \quad (19)$$

where $\mathbf{C}_{t_i}^F$ is the set of conclusions supported by t_i failing.

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 mutually supported by these two test outcomes. This area of mutual support indicates that the test outcomes are inherently non-conflicting, thus indicating we can deny the presence of conflict in the diagnostic process. Therefore, we can compute the relative denial of conflict between two test outcomes as follows:

$$\bar{\chi}(val(t_i) = \text{FAIL} \wedge val(t_j) = \text{FAIL}) = \frac{|\mathbf{C}_{t_i}^F \cap \mathbf{C}_{t_j}^F|}{|\mathbf{C}_{t_i}^F \cup \mathbf{C}_{t_j}^F|}. \quad (20)$$

Individual values for s_u or d_u depend on the confidence in the test outcomes and can be computed as

$$s_u(val(t_i) \wedge val(t_j)) = cf_{t_i} cf_{t_j} \chi(val(t_i) \wedge val(t_j)) \quad (21)$$

$$d_u(val(t_i) \wedge val(t_j)) = cf_{t_i} cf_{t_j} \bar{\chi}(val(t_i) \wedge val(t_j)) \quad (22)$$

As tests are evaluated, we accumulate support or denial for the unanticipated result in a similar fashion to equations 14 and 15 except that a single test outcome can cause several new “events” to be added. Formally, we perform the accumulation as follows:

$$\tilde{s}_u(\tau) = \left(\bigoplus_{i=1}^{\tau-1} s_u(val(t_i) \wedge val(t_\tau)) \right) \oplus \tilde{s}_u(\tau-1) \quad (23)$$

$$\tilde{d}_u(\tau) = \left(\bigoplus_{i=1}^{\tau-1} d_u(val(t_i) \wedge val(t_\tau)) \right) \oplus \tilde{d}_u(\tau-1) \quad (24)$$

where \oplus denotes combination as defined in equations 14 and 15. 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.

7. INTERPRETING CONFLICT

The previous sections provided algorithms for system diagnosis in the presence of uncertainty and conflict. As we discussed in the introduction, drawing conclusions from uncertain, but consistent test information is relatively straightforward. In this section, we will focus on the problem of interpreting conflicting test results. We begin by pointing out some basic assumptions for making a diagnosis in which conflict might arise.

First, we will limit our discussion to interpreting test results that either pass or fail. Note that this is not really a limiting assumption since all tests can be reduced to a set of binary outcome tests. Further, the algorithms we provided earlier will work with multiple-outcome tests as well as binary tests. Second, we assume that the diagnostic system is focusing on identifying a single fault. This assumption will appear, initially, to be very restrictive; however, we will see that this assumption will be useful for interpreting conflict as an indicator of a multiple fault which, in turn, facilitates multiple fault diagnosis. Normal assumptions in papers on system diagnosis include that the model is correct and the test results (or at least their confidences) are correct. Since test error and model error are prime causes of conflict, we will not make these assumptions.

We believe that only three fundamental reasons exist that might lead to conflict.

1. An error occurred in testing or test recording
2. Multiple faults exist in the system.
3. An error exists in the knowledgebase or model

By providing a separate conclusion in the model for conflicting information, we provide a powerful mechanism for identifying if one of three situations exists. We point out, however, that independent analysis may be required to distinguish these potential sources of conflict in any particular instance. The following paragraphs describe approaches that have been used in actual diagnosis to identify causes of conflict.

7.1 Identifying Errors in Testing

In real systems, testing is rarely perfect. In software, it is almost impossible to keep track of system state or to track all of the ways values might be affected. Hardware systems are frequently subject to noise and other random events making interpretation of test results difficult, thus lowering confidence in what the tests are indicating. Digital testing that may not be as susceptible to noise problems still must contend with the effects of state. Finally, modern systems heavily depend on both hardware and software, and the interactions between hardware and software further compound the problem of managing test error.

A common approach to minimizing the probability of error in testing is to apply automatic test methods to the system. The thought is that automatic testing

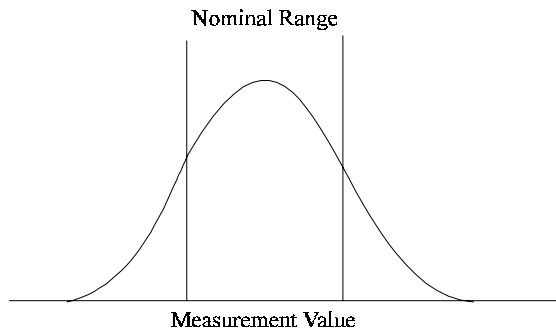


Figure 3. Nominal range for value.

takes much of the uncertainty out of testing since 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, introduce many more sources. Software must be written to run on the tester, and that software must also be tested. Until recently, much of the test software has been written from scratch with every new system to be tested leading to a high development cost and a high likelihood of repeatedly making the same mistakes in writing the tests. Finally, instrumentation used to apply test stimuli (or inputs) and interpret the response (or outputs) have physical limitations that can introduce error as well (Dill, 1995).

For example, suppose we are measuring a value in a system that may be subject to noise. In particular, the actual value (assuming nominal) may fit a normal distribution (assuming Gaussian noise), and the nominal value should appear within a specified range. (By the Central Limit Theorem, the Gaussian assumption will be valid when considering a large number of distributed variables, even if the individual variables are not Gaussian.) This situation is illustrated in Figure 3. Just as the system may have error in the output, the instrument measuring the output may also have error (based on the accuracy or precision of the instrument, among other factors). Therefore, we can model the nominal range for the instrument in a similar fashion (Figure 4). The problem arises when we overlay these curves. For example, is the value measured in Figure 5 nominal or not?

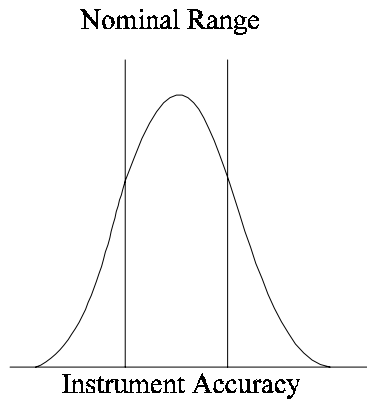


Figure 4. Nominal range for instrument.

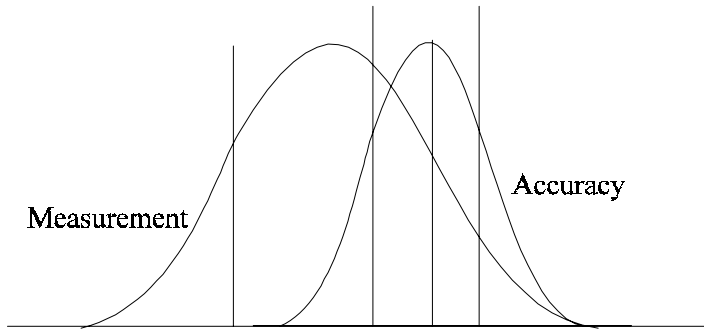


Figure 5. Uncertain pass/fail criteria.

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* into our test system. On the other hand, if the actual value falls outside of the nominal range and we declare that the test passed, we have introduced a *false assurance* into our test system. In

statistics, these errors are referred to as Type I and Type II errors. Which is which depends on whether our hypothesis is that the system 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. The larger the variance, the higher the probability of test error. When such sensitivity exists, several techniques are available to reduce the likelihood of error. One approach is to modify the decision boundaries (i.e., the tolerances) on the measurement. Unfortunately, tightening the bounds can lead to additional false alarms, and opening the bounds can lead to additional false assurance. A second approach is to take multiple measurements and either vote or take an average of the measurements. This approach increases the probability of an accurate measurement and interpretation as the number of measurements increases but can have a severe impact on testing efficiency. The approach also assumes that each measurement is independent of each other measurement, which is not always the case. Finally, one can take multiple, independent measurements at various points in the system and use the measurements as a consistency check on the other measurements.

In all three cases, when error exists 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. We prefer using the third approach to handling test error since it provides a means for identifying the conflict and for gathering supporting or denying evidence through testing additional. For example, we recently fielded a diagnostic system for testing the radar on an airplane. This radar used built-in test (BIT) which included a set of “accumulated” BIT tests. Accumulated BIT is analogous to the repeat polling methodology for limiting test error and consists of multiple evaluations of a test during flight. If the test fails, a counter in the BIT matrix is incremented. If the count exceeds some threshold, then the test failed; otherwise, it passed.

The diagnostic system we fielded used the Dempster-Shafer methodology for managing uncertainty and conflict. A preferred approach to handling the accumulated BIT would have been to treat each instance as a separate test with a low confidence. As the test is repeated, the confidence would accumulate with the counter and provide a more accurate indication of pass or fail during diagnosis. In this case, however, the diagnostic system only saw whether or not the threshold was exceeded, and a (perhaps inappropriately) high confidence was associated with the result. When the count is near the threshold, we might see the situation depicted in Figure 3 in which the support for unanticipated result rises.

As indicated earlier, determining the cause of conflict frequently requires *post mortem* analysis of the test results to determine the cause. In the absence of additional information, it is almost impossible to determine if the source of conflict was test error. In the case described above, test error becomes likely when considering the nature of the test itself. Since a “hard” threshold is applied to the test interpretation rather than applying either a “fuzzy” threshold or a confidence based on proximity to the threshold, the chances of not detecting a fault or reacting to false alarms increases. Knowledge of the nature of the test and the way the test results are fed to the diagnostic system facilitate this analysis.

7.2 Identifying Multiple Faults

The next step in managing conflict involves identifying the possibility of multiple faults being present in the system being tested. Multiple fault diagnosis is highly complex (deKleer, 1987; Sheppard & Simpson, 1994; Shakeri *et al.*, 1995). While many diagnostic systems exist that claim to diagnose multiple faults, and indeed it is possible to estimate most likely optimal faults fairly efficiently, the ability to correctly identify a multiple fault every time in a reasonable period of time is virtually impossible.

We use the conflict conclusion (i.e., *unanticipated result*) as an indication that a multiple fault may 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 problem of masked root cause faults and false failure indications). Therefore, if testing yields inconsistent results, one possible high probability cause is the presence of multiple faults.

Typically, multiple faults are identified and addressed in single fault systems using a sequential approach to diagnosis and repair (Shakeri *et al.*, 1995). In particular, first a single fault is identified and repaired. After the fault is removed from the system, the system undergoes additional testing to determine if another fault exists in the system. If so, it is identified and repaired as before. This process continues until all faults are identified and repaired.

This approach will work (albeit inefficiently) for all multiple faults in the system, with the exception of two. The first multiple fault situation in which “peeling” does not adequately identify and repair the system is when the *root cause* of system failure is masked by another, secondary fault caused by that root cause. When the root cause fault is masked by the secondary fault, both single fault isolation and any multiple fault isolation will identify only the secondary fault. This is a direct result of the masking effect. This in turn leads to ineffective

repair since repairing the secondary fault is futile. The secondary fault recurs when the system is reinitialized because of the root cause fault.

The second multiple fault situation in which “peeling” does not adequately identify and repair the system is when a *false failure indication* occurs. A false failure indication occurs when the symptoms of two or more faults are identical to another single fault. Once again, this leads to ineffective repair of the system since repairing the indicated single fault has no effect and the system is not restored to operational status. In other words, diagnosis will not have identified the faulty components, and maintenance leaves the system in the original failed state. Multiple failure diagnosis does not eliminate the problem, but the situation is improved. In particular, we find that with multiple failure diagnosis, we will be able to identify both the single fault that would be falsely indicated under single fault diagnosis, and the multiple fault. Unfortunately, we are unable to tell the difference between the two, and the two conclusions are considered to be *ambiguous*.

Generally, root cause situations can be identified through engineering analysis prior to fielding a diagnostic system, and common false failure indications can be tied to root cause situations. Therefore, it is still possible for single fault diagnosis, tied to good repair procedures, to address these problems. But what happens if multiple, independent faults exist in the system and we do not have the luxury of the time required to use the peeling strategy? Alternatively, what if certain faults simply cannot be removed from the system (e.g., a fault on a satellite in orbit)?

Conflict occurs when test results either directly or indirectly contradict each other. This 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. When conflict occurs, one possible cause is the presence of two independent multiple faults. Typically, when examining the ranked list of faults returned by these two techniques, the technician treats them in ambiguity (i.e., they look for one fault among the list). When conflict occurs, this can be used as a flag to the technician that more than one of the faults in the list may be present. Generally, the technician considers the faults in order of probability. This 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 is present.

The radar system described previously provided a simple example where a multiple fault appears to have occurred. In this case, the multiple fault was

benign in that both faults were contained within the same replaceable unit, and the maintenance personnel were only interested in fault isolating to the replaceable unit. Personnel at the next level of maintenance would have a problem when they attempted to repair further.

In this case, the problem was the presence of two antenna faults. Two independent BIT codes were obtained directly indicting two separate failure modes of the antenna. The diagnostic system noted the conflict and called for a separate “initiated” BIT (IBIT) to be run. The result was identification of the antenna with both faults identified and equally supported in the fault candidate list.

7.3 Identifying Modeling Errors

Modeling systems for diagnosis frequently entails capturing knowledge similar to traditional knowledge engineering tasks in expert system development. One common complaint with expert systems is that they are brittle, that is they are frequently incapable of handling situations in which the information obtained is inconsistent with the knowledge base. The availability of a special conclusion representing the presence of such an inconsistency, and the ability to further process the model in light of the inconsistency provides tremendous power in overcoming the brittleness of the traditional expert system.

If the test results are understood and no multiple faults exist in the system, then conflict may be caused by error in the model. The process of diagnostic modeling is extremely difficult and error prone. Identifying errors in the model is equally difficult. So far, the best approaches found for model verification have been based on analyzing logic characteristics of the models and performing fault insertion in the diagnostic strategies. Fault insertion is generally the most effective approach but can be time consuming and damaging to actual systems. Using fault insertion together with fault simulation eliminates the problem of potentially damaging the system; however, the verification problem now includes verifying that the simulation model is correct. Otherwise, the only assurance gained from fault insertion in the simulation model is that the diagnostic model represents the simulation, not the actual system.

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

indications can all indicate potential problems with the model. Details on using these characteristics in verification are provided in Simpson & Sheppard (1994).

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, with the *unanticipated result* included in the set of possible conclusions, a “flag” can be raised whenever a model error might have been encountered. As with test error and multiple fault diagnosis, if the model is error, it is likely at some point for a test result to be contrary to what is expected—in which case a conflict occurs and the *unanticipated result* gains support.

In the case of the radar system, we were able to find model errors since two types of models were available for comparison—a system-level model and multiple replaceable unit-level models. (While the two types of models were not derived independently, they offer alternative representations of the system.) When test results were processed through the model representing the transmitter, no conflict was encountered and a large ambiguity group consisting of wiring, the transmitter itself, and the computer power supply. When the same test results were run through the system-level model (the model of choice in the field since the replaceable unit did not need to be known prior to testing, 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.

When we noticed this discrepancy, we looked at the diagnostic log files for the two runs. In the case of the replaceable unit-level model, only one test were considered, and that test pointed directly at the fault in question. 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, included these four, depended on the common test in system-level model and all passed where the common test failed. Since each of these tests depended on the common test, when the common test failed, each of these additional tests should have failed as well. Since 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. The failing test determined whether the transmitter was transmitting by examining the least significant bit in a data word sent in a loopback test.

The more appropriate representation of this situation in the model would have used what is referred to as a “linked test outcome.” Information flow models assume “like” inferences. In other words, if a test fails, it is assumed test inferences drawn from that test will consist only of failing and if the test passes, test inferences drawn from that test will consist only of passing. But this is not a realistic assumption, so the information flow model permits declaration of special inference linkages—linked test outcomes—in which a passing test can result in other tests being inferred fail or unavailable and a failing test can result in other tests being inferred pass or unavailable. The skipped test in the radar transmitter model should have been linked to the each of the tests that would have caused it to be skipped such that failure of those tests would cause an inference of the skipped test being unavailable. This was not used in the model resulting in an error. In fact, it was found that a “hard” dependency existed between the collector test (the skipped test) and the communications test in the system model and *no* dependency existed at all in the replaceable unit-level test. Both models were in error but for different reasons.

8. SUMMARY

The process of diagnostic modeling and diagnostic testing is complex and often leads to conflicting information. Traditional tree-based approaches may miss apparent conflicts and provide reduced diagnostic accuracy. Whether interpreting uncertain test results, handling multiple faults, or contending with modeling error, the diagnostics used should support gathering information capable of identifying conflict and assessing the cause of the conflict. We presented two approaches for identifying conflict within the framework of the information flow model and provided several examples of how one can use the identification of conflict to improve the diagnostics of the system.

Uncertain and inexact testing can lead to conflict in diagnosis in that the outcome of the test as measured may not accurately reflect the system being tested. This can occur when the error range of the test instrument compared to the tolerance of the expected test results overlap significantly. Identifying when this situation exists involves a detailed understanding of the failure modes detected by the test and the error range of the instrument relative to the test. The occurrence

of conflict in testing can indicate the need for closer examination of the test results by identifying that unexpected test results have occurred.

In addition to the ability to raise the concern of problem tests, detecting and analyzing the presence of conflict can be useful in multiple fault diagnosis when the diagnostic procedure begins with the assumption of a single fault. When multiple faults exist, test results can be consistent with one of the faults but inconsistent with the other. Since the process of multiple fault diagnosis is computationally complex, using the presence of conflict to flag the need for checking for multiple faults is a reasonable and computationally efficient approach to take.

Finally, since modeling and knowledge engineering is a complex and error prone task, the diagnostics can be used to assist modeling by identifying when inconsistent or illogical 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 an inadequacy 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, an analyst can localize the potential cause of the conflict and identify possible problems in the model.

To date, little discussion has occurred on the positive role of conflict in system test and diagnosis. Conflict has always been regarded as something to be avoided. But conflict can provide valuable information about the tests, the diagnostics, and the system being tested. In this paper, we attempted to describe an approach for capturing and quantifying the amount of conflict encountered in testing and to describe approaches to using the conflict to benefit the diagnostic process, thus leading to more robust overall system diagnostics.

9. ACKNOWLEDGMENTS

This paper describes the results of work that has been performed over several years, and the authors have received input and guidance from several people to improve the techniques and the paper itself. We would like to 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.

10. REFERENCES

- Cantone, R. And P. Caserta. 1988. "Evaluating the Economical Impact of Expert System Fault Diagnosis Systems: The I-CAT Experience," *Proceedings of the 3rd IEEE International Symposium on Intelligent Control*, Los Alamitos, California: IEEE Computer Society Press.
- Davis, R. 1984. "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence*, 24:347-410.
- deKleer, J. 1987. "Diagnosing Multiple Faults," *Artificial Intelligence*, 28:163-196.
- Dempster, A. P. 1968. "A Generalization of Bayesian Inference," *Journal of the Royal Statistical Society*, Series B, pp. 205-247.
- Dill, H. 1994. "Diagnostic Inference Model Error Sources," *Proceedings of AUTOTESTCON*, New York: IEEE Press, pp. 391-397.
- Dretske, F. I. 1982. *Knowledge and the Flow of Information*, Cambridge, Massachusetts: The MIT Press.
- Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*, San Mateo, California: Morgan Kaufmann Publishers.
- Peng, Y. and J. Reggia. 1990. *Abductive Inference Models for Diagnostic Problem Solving*, New York: Springer-Verlag.
- Pople, H. E. 1977. "The Formulation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning," *Proceedings of the 5th International Conference on Artificial Intelligence*, pp. 1030-1037.
- Quinlan, J. R. 1986. "The Induction of Decision Trees," *Machine Learning*, Vol. 1, pp. 81-106.
- Shafer, G. 1976. *A Mathematical Theory of Evidence*, Princeton, New Jersey: Princeton University Press.
- Shakeri, M., K. R. Pattipati, V. Raghavan, A. Patterson-Hine, and T. Kell. 1995. "Sequential Test Strategies for Multiple Fault Isolation," *Proceedings of AUTOTESTCON*, New York: IEEE Press, pp. 512-527.
- Shannon, C. E. 1948. "A Mathematical Theory of Communications," *Bell Systems Technical Journal*, Vol. 27, pp. 379-423.
- Sheppard, J. W. 1996. "Maintaining Diagnostic Truth with Information Flow Models," *Proceedings of AUTOTESTCON*, New York: IEEE Press, to appear.
- Sheppard, J. W. and W. R. Simpson, 1994. "Multiple Failure Diagnosis," *Proceedings of AUTOTESTCON*, New York: IEEE Press, pp. 381-389.
- Shortliffe, E. H. 1976. *Computer Based Medical Consultations: MYCIN*, New York: American Elsevier.
- Simpson, W. R. and J. W. Sheppard. 1994. *System Test and Diagnosis*, Norwell, Massachusetts: Kluwer Academic Publishers.
- Wilmering, T. J. 1992. "AutoTEST: A Second-Generation Expert System Approach to Testability Analysis," *Proceedings of the ATE and Instrumentation Conference West*, pp. 141-152.