Smart Diagnostics Pilot: Design Dynamics

John Mark Agosta, Thomas Gardos
Intel Corporation
{john.m.agosta, thomas.r.gardos}@intel.com

The *Smart Diagnostics* project's purpose is to measure the value of knowledge-based Bayes networks for automating assistance to semiconductor plant technicians. Such an evaluation is a necessary step when introducing new technology in an industrial environment. To carry this out we have built a diagnostic model for fault isolation of machine failures of a critical tool used in Intel's manufacturing, and we have constructed a custom web-based application with which to field the model. [1,2] Having recently completed the application, our next step will be to use the application to observe user's behavior, to validate and evaluate the model and thus gain evidence for the value of this approach. By user behavior we refer specifically to the user's choice of tests and repairs upon receiving an indication that a machine has failed. Our premise is a natural one for such a model, that by presenting to the user an optimal sequence of test and observation operations based on information gain we can improve their performance in terms of time to reach an acceptable diagnosis. Given that this premise can be experimentally substantiated, the next question will be whether this makes a difference to their current practice on the efficacy of troubleshooting with a web-based assistant, a question that turns on many currently unconsidered aspects of the software, such as its value in fostering interactions among users.

In this paper we present our experience designing the test application and the experiment it will support. It might not come as a surprise that the static aspects of the application design, e.g. the layout of forms, has been easy to address compared to the dynamic aspects, in particular the presentation and analysis of the diagnostic sequence. The application presentation is designed to be modeless: The diagnostic sequence is determined by the user. She is given the freedom to change any diagnostic input at any time in any order, and see the consequences in the diagnostic outputs. By its nature the application keeps no diagnostic state beyond the state of the user inputs, so that resetting inputs to a previous state results in identical outputs to that previous state.

The application is implemented as a set of test tabs offering different browsing strategies, by test category, by name, or by information gain ranking. When the user submits new test outcomes, the recommended list of repairs is updated. We are interested in the user's strategy when selecting tests, especially if she benefits from using the information gain ranking of recommended tests. The application logs a user's choice of strategy (e.g. the tab selected) and sequence of tests. A screenshot of the application is shown in Figure 2.

The application software is intended eventually to be fielded as a production application. The logging instrumentation in this version is there to record the sequence of actions in a user's session. In the fielded application this logging will be kept in place where it is expected to collect closed cases that can be used for model improvements.

Some recent studies [3, 4] have presented results of diagnostic Bayes network evaluations. The two cases cited both had the advantage of comparing their Bayes network performance against a "gold standard" set of cases. In this work, we attempt to measure instead the usefulness of the application and model, with the presumption, possibly refutable, that the model is correct.

Experimental Approach

Our entire experimental plan consists of two phases:

1) Model Validation: Test validity of the model "offline" using expert technicians who were not involved in elicitation of the model. The model is run as a simulation against itself and compared with a user running against the simulation.

2) Model Evaluation: Pilot the model in actual manufacturing operations using a mature web interface. The model's recommendations would be compared with actual user behavior. This evaluation tool may turn into the basis for an actual fielded system.

Both would be interactive experiments with technicians as users, during which the session logs we collect could be used to compare actual the diagnostic session recommendations with those predicted by the model. In this paper we concentrate on validation.

Simulation algorithm

BEGIN

1. Select a target variable, cycling through the list. (The results can be weighted by the target priors later if desired.)

2. Instantiate the target, and compute the test marginals, conditioned on the target fault. Choose settings (outcomes) for the primary indicator tests by Monte Carlo with the computed test marginals. (At the same time, the outcomes of the recommendable tests can be computed, but these will not be revealed until the user chooses tests.)

3. Present the primary indicator settings to the user,

DO {

4. Present the user the rankings of the recommended tests. The simulation determines outcome of test through most likely outcome (MAP) or Monte Carlo simulation.

5. The user chooses one or more tests to "perform" (thus their outcomes are revealed to the user.)

6. The tests are instantiated to the values chosen by the simulation, and the fault rankings and recommended test rankings are updated.

7. The new fault rankings are revealed to the user.

} UNTIL (8. User chooses a component as the most likely repair **);**

END

Figure1: Validation pseudo code

Validation use cases.

In the validation phase, the tool presents a hypothetical session, starting by choosing a simulated fault—unknown to the user—and initiating the session by presenting predicted primary indications to the user. For each set of observations the user enters, the simulated response by the model is revealed. The model recommendations are then recomputed based on the revealed observations. This is detailed in Figure 1.

There are three scenarios to be compared in validation experiments, indexed by who is the user (man or machine) and if recommendations are available:

1. *Automated simulation.* We simulate the observation of recommended tests by running the model against itself; (The "user" is a second copy of the model.) one copy of the model predicting observation outcomes for a fault, and the other using the predictions to generate test recommendations, in ignorance of the simulated fault.
2. *Model guided simulation.* The user has the benefit of observing the recommendations of the model at each stage before selecting tests. The test result simulation is done as in scenario 1. The tool also records user's explanations for when they make a choice distinct from the model recommendation.
3. *Blind simulation.* Like scenario 2, but the user explores the diagnostic sequences in the interactive web environment without the recommendations of the model. The log records the match between user choices and model recommendations at each step.
4. *(The case of an unguided automated simulation is uninteresting, but could also be measured.)*

There are two minor variants of the simulation method to consider, based on different interpretations of test outcome randomization. In one method, the entire list of test outcomes are simulated after the fault is instantiated. This appears reasonable since the test outcomes should be determined only by the fault, and not be the instantiated values of other tests made previously in the session. However, in actuality in a diagnosis, tests will be instantiated at each step of the session. A clear advantage to

generating just one randomization is that cases with the same test randomization can be analyzed pair-wise, e.g. user versus automated simulation, and the variance in the outcomes reduced.

The methods by which test outcomes are "simulated" can also vary. If the most likely value for each test is used as the instantiated value, then every run of that fault will follow the same test tree with different branch orderings. Instead if test outcomes are selected by Monte Carlo based on the test marginal given the selected fault, then the full diagnostic tree is explored. The latter method appears more realistic; however it does increase the experiment variance, and experience with our model shows many cases where the simulation does not converge on the chosen fault. Clearly there are open questions about how the simulation is best performed.

Logging:

During this entire session we capture user responses, as currently logged by the web application. And if the user selects either test or fault that is not top ranked, then the application will prompt the user to explain why other test or fault was selected.

Evaluation use cases.

The evaluation use cases are not yet determined, but would incorporate features similar to the model guided simulation, with the actual machine maintenance session taking the place of the simulation. In both validation and evaluation experiments, the variety of trials we can run will be limited practically by the number of technicians who specialize in this area. Running validation cases is relatively quick, so that the number of case samples for a given technician is constrained only by the variety that the model can present and perhaps by the user's attention span. Technicians' level of expertise is known, so that the sessions can be labeled as expert or novice user sessions. As for evaluation use cases, the number of unscheduled troubleshooting sessions in this area within Manufacturing are so numerous that a month should be adequate to collect an informative set of cases.

Experiment Description

Our primary question is to determine whether users benefit from the model's sequence of test recommendations. A simple metric for this is whether the session results in the correct diagnosis, and given it does, how few tests were necessary to reach the diagnosis. If expert users do not accept the model's choice of diagnosis, it is evidence that the model is incorrect. If the choice of diagnoses agree, we propose to compare the effect of the model's recommendations among the three scenarios. If the model itself is accurate and the model's recommendations (including the manner in which the recommendations are presented to the user) are valuable, we'd expect the model-guided simulation to dominate the blind simulation results, and be bounded by the automated simulation results.

An interesting case arises if the model itself is not accurate. This may be apparent from expert users' results. If blind simulation with expert users approaches or exceeds the model's automated simulation results, then the model itself is called into question. Obviously if the model is inaccurate, the recommended tests would not be dependable, and the experiment could serve to solicit model improvements from users.

Experiment Data Analysis

We will have logs from three sets of scenarios, corresponding to the validation use cases:

1. The "optimal" diagnostic sequences as computed by the diagnostic model for the simulated fault and test outcomes. There is no human in the loop in this case.
2. The users' troubleshooting sequences with recommendations from the diagnostic model.
3. The users' troubleshooting sequences without recommendations from the diagnostic model.

Use case 1 will be our baseline for the experiment analysis. Note that with Monte Carlo simulation of the test outcomes, even in this use case we will have multiple troubleshooting sequences for the same fault based on the different test outcomes.

In use case 2, although we prioritize the tests based on information gain and compute posterior probabilities for the faults, we do not force the user to perform the highest ranked tests or choose the most likely faults. If they did, the outcome would be the same as use case 1. In cases where the user strays from the recommended test sequence we will pay special attention to whether the user's sequence arrived at the fault in fewer steps (a scenario which may occur with expert diagnosticians) or whether they took more steps to arrive at the fault. In the latter case, the user would have been better off following the model's recommendations. As for the instances where the expert's performance exceeds the model's, these instances can be reserved for subsequent updating of the model.

For case 1 then, we will present summary statistics for the number of diagnostic steps for each fault. We will present an overall summary statistic for all the faults by weighting the summary statistics by the fault priors. We will also present which tests were included in the collection of troubleshooting sequences for each fault and show a histogram of the outcomes of each of the tests.

We will similarly calculate the summary statistics for cases 2 and 3. To verify experiment coverage we will ensure that the ratio of test outcomes match the test outcome ratios for case 1. If they do not, it would indicate test outcome combinations that weren't well represented in the experiments.

Once the summary statistics are calculated from all the cases we will then calculate *efficiency ratios*. For example, we can compare the average ratio of diagnostic steps an expert user takes in case 2, versus the blind simulation in case 1 as

$$Avg[(n_2 \mid expert) / n_1] \quad (1)$$

where n_2 is the number of steps in case 2 and n_1 is the number of steps in case 1.

Similarly for the novice user we would calculate

$$Avg[(n_2 \mid novice) / n_1]. \quad (2)$$

Presumably ratio (1) will be close to unity, while ratio (2) will be greater than 1.

Similarly we would expect that

$$Avg[(n_3 \mid novice) / (n_2 \mid novice)] \quad (3)$$

where n_3 is the number of steps in case 3. We would expect this to be greater than one, indicating the diagnostic model improved the novice diagnostician's performance.

Design questions

We present the application and experimental designs as a set of open questions in the spirit of soliciting comment and advice from the research community. These questions will have a bearing on completion of the software tool and the analysis of the data collected. Some additional open questions are the granularity with which we log user sessions, the ability to query and capture user feedback on why recommended tests and faults were or were not selected, and how to aggregate user responses to quantify model performance.

Contextual factors affecting evaluation outcomes

We are also interested in larger questions about the use of such a tool on the behavior of users, given it is successful according to the criteria we've applied. On the one hand, it is likely that such a tool's value could be as a training tool. Novice users may refer to it to run hypothetical cases by which to test their understanding of the system. On the other hand, it may be seen as a "de-skilling" tool, where users rely on the tool to the point where they forget how to reason diagnostically without it!

The context in which the software will be used contributes factors and uncertainties affecting the evaluation outcomes. Considering that estimating time to repair based on tests taken is the outcome of interest, this outcome is subject to these other factors that are part of the "use case" for technician repair:

1. Technician availability for "hands-on" labor
2. Engineer availability for advice and fault escalation
3. Material resource availability: replacement parts, tools, and shipping.
4. Higher priority demands on availability of human and material resource. Our study borrows the common impression that reducing unscheduled repair time is the best

way to meet the larger goals of the plant. Obviously this is not the only goal of the plant.

5. Competing organizational procedures, not necessarily justified as higher goals, such as safety, training, reporting, and scheduled maintenance requirements. Related to this, the competing software application needs, for asset management, logging, and "pass down" (informing the next shift what to do)
6. Competing standard practices for measuring performance.

Our approach is not to ignore these factors, but to characterize the model's performance in terms that best describe its advantage, then see how it fits into the larger context. Stated simply, the main contribution we see the model is capable of making is captured by measuring completion time.

Production software serves a "basket" of purposes, and our user's expectations make it clear that a tool that just reduces number of diagnostic steps is only part of what the user needs. Software that is already in place logs users' steps, and tracks machine performance to meet several organizational needs. In essence we see the model as one module, to integrate with, and to fill a functional gap in existing systems.

Another area of investigation not considered here is the design of tools for building models, and for incremental learning from past successful trouble-shooting cases. Our impression is that the users who would build and maintain models would be the engineers responsible for setup and documentation of equipment function, a group that is distinct from the technicians who carry out the actual repairs.

References:

[1] Agosta, J.M., T. Gardos, *Bayes Network "Smart" Diagnostics,* Intel Technology Journal. http://developer.intel.com/technology/itj/2004/volume08issue04/art10_bayesnetwork/p01_abstract.htm (November 2004).

[2] Agosta, J.M., T. Gardos, *Wafer Fab Diagnostics,* presentation at UAI 2004 Bayes Applications Workshop, Banff, Canada.

[3] Fraser, H.S.F., W.J. Long, S. Naimi, *Evaluation of a Cardiac Diagnostic Program in a Typical Clinical Setting* , J Am Med Inform Assoc. 2003;10:373–381.

[4] Oni'sko, A., M. J. Druzdzel, and H. Wasyluk. *A Bayesian network model for diagnosis of liver disorders*. In Proceedings of the Eleventh Conference on Biocybernetics and Biomedical Engineering, volume 2, pages 842--846, Warszawa, Poland, December 2--4 1999.

Bios:

John Mark Agosta develops diagnostic, optimization and statistical models to improve Intel products and processes. Previously he has worked for Edify Corporation, Knowledge Industries, and at SRI International, building diagnostic and planning models for medical, equipment, and emergency response applications. Agosta received his Ph. D. in the Stanford University's Engineering-Economic Systems Department (now Management Science and Engineering) in 1991.

Thomas Gardos is senior researcher and program manager in Intel's IT Research. He is currently investigating machine learning approaches to enterprise manageability and exploring Bayes network based diagnostic systems. Previously he was technical lead and manager of Intel´s telephony and Internet streaming video coding team. He was Intel´s representative to the ITU-T and MPEG video standards committees and has taught video and image coding at universities, conferences and industry short courses. He holds 13 patents in this field. Tom received his Ph.D. in EE from the Georgia Institute of Technology in 1993.

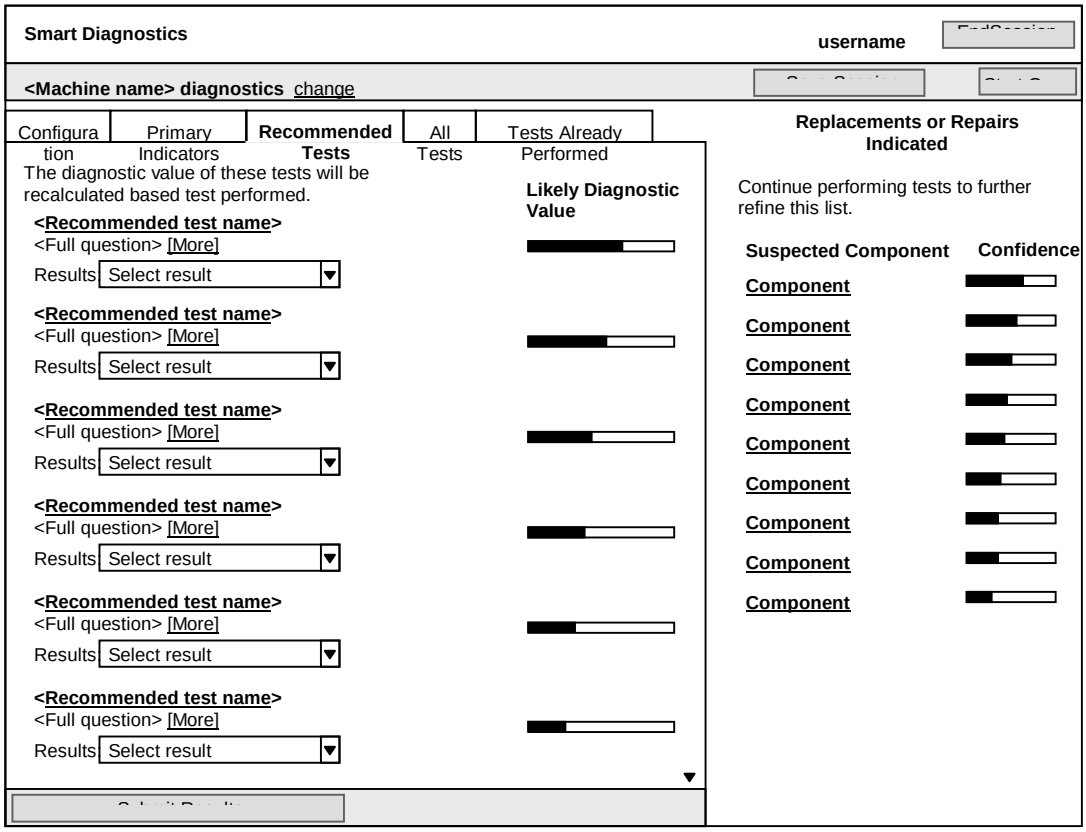

Figure 2: Example of a web-page “wireframe” design, as built in consultation with the HCI designer.