Published in Proceedings of the IEEE Conference on Artificial Intelligence for Applications, (CAIA'93), Orlando, Florida, March 1993, pp. 121-127. (Best paper award.)

Categorization-Based Diagnostic Problem Solving in the VLSI Design Domain

Amir Hekmatpour
IBM
Mail Code B21/975-1
Essex Junction, Vermont 05452

Charles Elkan
Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093-0114

Abstract

This paper introduces a novel architecture for expert systems performing diagnostic problem-solving. The architecture is called "categorization-based" because problems are solved by relating them to relevant categories of known cases using deep knowledge about the application domain. The effectiveness of the architecture has been validated by an implemented expert system called CHATKB*, which does troubleshooting for users of IBM computer-aided VLSI design tools. Experiments show that* CHATKB *provides advice of the same quality as human experts.*

The deep, general knowledge in CHATKB *about the VLSI design domain was developed by studying versions of* SPICE*. The success of* CHATKB *now shows that this knowledge is applicable to quite different VLSI design tools. With the categorization-based architecture, it therefore provides a sound, transferable basis for creating expert systems to help the users of most VLSI design tools.*

1 Introduction

CHATKB is an implemented and deployed expert system which assists users of IBM tools for computer-aided VLSI design in overcoming errors of all types and origins. CHATKB solves a problem by categorizing it using a general framework for understanding errors in VLSI design, and then retrieving relevant previous cases from a library of about 250 cases derived from about 400 descriptions of episodes of error analysis and recovery directed by human experts. The architecture of CHATKB thus has both a heuristic classification aspect [3] and a case-based aspect [24].

The deep, general knowledge about the VLSI design domain possessed by CHATKB is an extension of a framework for understanding errors in the VLSI design process developed in previous work by studying three versions of SPICE. Work on CHATKB now shows that the framework is applicable to completely different VLSI design tools. This deep knowledge and the categorization-based architecture jointly constitute a well-defined, transferable methodology for quickly building expert systems to help the users of most VLSI design tools.

This paper is organized as follows. Section 2 describes the starting point of work on CHATKB, a database named CHAT of email messages between users and developers concerning problems with a variety of IBM internal VLSI design tools. Section 3 discusses the framework for analyzing errors in the VLSI design process. Section 4 describes the knowledge engineering that produced the CHATKB knowledge base from the framework and the free-format database of problem-solving episodes in CHAT, while Section 5 explains the inference algorithm of CHATKB and Section 6 describes its implementation and user interface. Finally, Section 7 presents the results of experiments showing that CHATKB provides advice of the same quality as human experts, and Section 8 evaluates the contributions of our project.

2 The CHAT problem database

CHAT (Customer Help At Terminal) is a repository for user complaints and misunderstandings concerning a set of tools for VLSI computer-aided design supported by the Design Automation group at the Burlington, Vermont IBM site. Design engineers report problems by email in an open-ended, free format. They can also search a database of all previously reported problems to see if any similar problem has been resolved, and they can answer recent problems if they wish. The designer who reports a problem can try any suggested solution and report its success or failure.[1]

The CHAT database contains all email messages sent between designers, design tool support personnel, a logistics group, and the CHAT support group. In addition, the database can contain additional relevant files such as input and output files and files of technology rules.

[1] Each CHAT problem is tagged as one of *open*, *accepted*, *pending*, or *closed* (a few more codes actually exist also). Once a problem is reported (*opened*) on CHAT, it is manually routed to the appropriate support personnel. When these people agree that a real problem exists, the status of the problem is raised to *accepted*. When a solution to the problem is proposed, its status becomes *pending*, and when the person reporting the problem accepts the solution, the problem is *closed*.

Problem Area	*Number of Cases*
Timing	141
Physical Design	102
Design Capture	46
Logic Synthesis	35
Simulation	26
General	19
Test	15
VIDAS	13
View	2
Total VLSI Design	399
CHAT	10
Total CHAT cases	409

Table 1: Distribution of CHAT problem reports.

In order to be able to analyze CHAT cases in depth without contaminating the database, we backed up the database when it was just over eight months old, in November 1991. At this point it contained 409 cases, and was growing at an average rate of 30 cases per month. Table 1 shows that about 35% of the cases concerned timing, and of these, 77% were related to one design tool named ETE (Early Timing Estimator). Since over a quarter of all problems concerned this one tool, we decided to concentrate on it first.[2]

There are three main reasons why ETE gives rise to more problems than any other tool. First, designers are not as familiar with ETE as with other VLSI design tools. Although the first timing correction tools date from the early 1980s, compared to other aspect of VLSI design such as logic design, simulation, placement, wiring and physical layout, timing correction is novel. Second, timing correction is not as automated as other tasks in VLSI design. ETE tries to reduce (by using faster elements) or increase (by adding buffers) the delay through a signal path on a chip. In some cases ETE can also combine paths or portions of paths. Design changes beyond what a user expects can happen easily, so considerable user intervention is needed. Third, the Burlington IBM site has corporate-wide responsibility for support and maintenance of ETE and its technology rules. A large number of problems come directly to Burlington, and all questions not resolved locally end up in Burlington.

3 A framework for analyzing errors in the VLSI design process

In preparation for work on CHATKB, we studied existing theories of error closely [17, 20, 16, 21, 8]. An attempt to map these to the domain of VLSI design revealed some limitations and mismatches. The limitations stem principally from the fact that existing theories focus on explaining errors made by a human operating a well-defined and debugged system, such as the control room of a nuclear power plant. Although they may be very complex, systems of this nature provide a single, consistent user interface and a continuous interaction environment. VLSI design tools provide a very different environment.

The VLSI design process involves many design activities supported by a wide variety of independent design tools. These tools and their corresponding design methodologies are typically developed by different vendors and they may run on different operating systems and hardware. Each design tool has its own look-and-feel and interaction style: some tools can only be run in interactive mode, some only in batch mode, and some in both modes. Tools vary also in how errors are reported: some tools crash gracefully and produce meaningful feedback, while others produce tons of core dump.

Perhaps most important, different tools rely on widely different task models. A task model is a decomposition of a task into standard subtasks. For the same application, the task models used by different design tools can vary in which subtasks are identified, and in how these are organized. The task model assumed by a particular VLSI design tool depends on the nature of the application, traditions associated with the application, and choices made by the creator of the tool. Some tools just perform a predefined set of operations on their input. With these tools there is little scope for user initiatives or for knowledge-based automatic error recovery. Other tools require considerable user intervention, but each choice made by the user is simple: the organization of subtasks is shallow and narrow. Other tools require considerable conscious planning and thought from a user; their task model includes deliberate trial and error and backtracking.

Another consequence of the fact that existing theories of error concentrate on modelling errors made by humans working in a well-defined environment is that these theories pay most attention to modelling the cognitive aspects of errors. They pay little attention to other types of errors or to the process of recovering from errors.

Given the limitations just discussed of existing models, we developed a new framework for understanding how errors occur in VLSI design and how recovery happens. This framework accommodates varying task models, and applies to all types of errors, including system failures and design specification errors.

The first part of the framework is a generic, hierarchical scheme for classifying episodes in which errors happen, are interpreted, and are recovered from (error episodes). The basic categories of this taxonomy are tasks, conditions, symptoms, and initiatives. A task is a phase in the VLSI design process, and a condition is some aspect of the state of the current task. A symptom is an observable phenomenon such as an

[2] The current version of CHATKB can also help users of the other tools supported by CHAT, but the quality of coverage for them is not as good. Since the methodology for acquiring knowledge is tool-independent, we do not anticipate problems in improving coverage later.

error message or an incorrect output. Finally, an initiative is an action that changes some conditions.

The second part of the framework is a collection of guidelines for analyzing transcripts of episodes of error interpretation and recovery. Error interpretation is the process of identifying which underlying conditions hold given facts about the current task and symptoms, while error recovery is the process of choosing initiatives that change conditions in a desired way. These guidelines deal with identifying and ranking the observability of conditions, their relevance to symptoms, and generating plans of initiative and recovery.

The initial version of this framework for understanding error analysis and recovery was developed by analyzing three versions of SPICE, a well-known VLSI circuit simulator. Over 100 episodes of error diagnosis and recovery were generated, involving errors of all sources: user, design, system, and interface. These cases were analyzed using the guidelines developed, and generalized as far as possible to include all combinations of symptoms and conditions sharing the same diagnosis. Over 95% of the cases could be classified using the taxonomy developed. Those cases that could not be classified were compound or transient problems which could not be solved at all.

An important part of the new framework for understanding VLSI design error episodes is a three-dimensional graphical classification tool called TC$\mu\rho$ (pronounced tea-cup). This has proved very helpful in conceptualizing the task-symptom-condition space. TC$\mu\rho$ is an acronym for Task, Condition, μ (condition density), and ρ (symptom density). A TC$\mu\rho$ diagram is a graphical classification of correspondences between tasks, conditions, and symptoms. In a TC$\mu\rho$ diagram, conditions are plotted on the x axis versus design tasks on the y axis (see Figure 1). For each intersection of a task and a condition, all corresponding known symptoms are displayed as directed arrows with their height indicating their symptom density. This is the degree to which a symptom (for example an error message) corresponds uniquely to a condition, during a specific task. More formally, symptom density

$$\rho = Pr(\text{ condition } \wedge \text{ task} \mid \text{symptom }).$$

The degree of observability

$$\mu = Pr(\exists \text{ symptom} \mid \text{condition } \wedge \text{ task })$$

of a condition during a certain task is indicated by the darkened area of the square at which the task and condition intersect.

TC$\mu\rho$ diagrams make the dominant symptoms and most and least observable conditions graphically evident for each VLSI design task and subtask. Although generating TC$\mu\rho$ diagrams is a time-consuming manual process, they are very valuable in the early stages of doing a task analysis for a design tool and in case generation and analysis. However, TC$\mu\rho$ diagrams do not allow us to represent knowledge about initiatives and recovery procedures. We plan in future work to investigate the possibility of including knowledge about recovery initiatives with each symptom vector.

Figure 1: An example TC$\mu\rho$ diagram.

4 Knowledge engineering

The first stage of knowledge acquisition for CHATKB was to extract, organize, and classify all the important information in the 399 natural language CHAT transcripts of error episodes. This work followed the guidelines and used the taxonomy discussed above. The guidelines proved to be very helpful in synthesizing coherent episode scripts from scattered descriptions of events and conditions. This classification process revealed about a dozen general problem classes, each corresponding to a general class of appropriate recovery initiatives. These results are similar to our experience with versions of SPICE, which is encouraging, since at first each CHAT case appeared unique. We are sure anyone else would have initially had the same feeling from looking at the 20 to 30 pages of email messages associated with the typical case.

The next stage of knowledge acquisition was to identify and acquire knowledge not explicit in the CHAT transcripts. This was done in a consistent and efficient way thanks to the use of a standard frame structure for representing cases, the so-called UPERIT (User, Problem, Error, Recovery, Initiative Template) template. This template is an expanded version of a template that we used in our earlier study of SPICE. Filling in the UPERIT template for each case is the most time-consuming phase of knowledge engineering, because cases are in fact generalized here. Each case is analyzed to identify whether it has any significance beyond its own facts and whether it applies to a wider range of problem situations.

Approximately 10% of CHAT cases could not be mapped onto UPERIT templates or were only partially mapped. Some of these cases were composite problems (chain reactions where the original problem could not be identified), while others were due to transient glitches in the system, or might have been due to hardware rather than software failures. In addition, some of these cases were still under investigation at the time the CHAT database was backed up.

Each field of a UPERIT template is classified as surface or deep. A surface feature is a low-level feature, whose value is inexpensive to obtain [23]. Identifying deep features and finding their values, on the other hand, requires domain knowledge and is time-consuming. On the whole, symptom and initiative fields are surface, while task and condition fields are deep. Case features are also classified as predictive or non-predictive, as in CYRUS [11] and UNIMEM [13].

A distinctive feature of CHATKB is that cases were assessed for similarity and merged if possible during knowledge acquisition. During case generation (mapping free-format CHAT cases to UPERIT templates) predictive fields were used to identify similarity, and thus to fill in values for deep fields based on the similarity of the case at hand and previous cases. Moreover, similar cases were combined when possible into a single case, with non-identical features accommodated as multiple values.

Similarity assessment is loosely based on the nearest neighbor metric [5] as extended in [1]. This metric says that the similarity between two cases is proportional to the number of matching predictive features that are relevant for the two cases. Importantly, different features are classified as relevant for different cases. Case retrieval and ranking use the same formula for similarity assessment.

5 Inference in CHATKB

Cases in the CHATKB knowledge base are hierarchically grouped into classes. Given a partially completed template for a new case, the CHATKB inference algorithm first identifies a class of relevant cases, then a subclass of this class, and finally the specific most similar case(s). This section first explains the hierarchical organization of knowledge, and then presents the case retrieval algorithm.

The CHATKB knowledge base is modularized, with three main levels of knowledge. At the top, one knowledge module captures the overall conceptual organization of the VLSI computer-aided design domain. Intermediate level knowledge modules concern individual VLSI design tools, while knowledge at the lowest level consists of UPERIT templates.

What the top-level module specifically contains is knowledge about how different tools and stages of the VLSI design process are related to each other. We follow [10] in partitioning the design process into algorithm, architectural, structural, functional, logical, transistor, technology, physical, package, and fabrication stages. This module includes, for example, the knowledge that ETE is involved in the tasks of functional and logical design.

The top and intermediate level knowledge modules are trees (actually, directed acyclic graphs). Near the top, "timing" is one of the subbranches of the "functional" and "logical" branches; ETE and other tools used during timing are stems of the the "timing" subbranch. For simple tools, the corresponding stems are leaves, while for tools like ETE that use other tools, the stems have their own branches.

Each leaf of the top-level module is connected to a single intermediate level module, which contains knowledge about how the corresponding tool works. For example the module for ETE represents the relationships between input files, timing rules, option files, timing specification files, output files, and other tools called by ETE. Each tool module leaf is associated with a set of UPERIT templates. For example the terminal node "timing specification problems" in the module for ETE is associated with templates for problems concerning specification syntax and specification limits.

Each node in the top-level and tool knowledge modules has an attribute called node strength, obtained from historical data and the opinions of experts, which represents the degree of *a priori* certainty that the node is responsible for the condition(s) described at its parent node. Node strengths make each knowledge module resemble the causal inference network used in [12] where the node for each condition is linked to nodes for other conditions caused by it, with associated probabilities.

Each node also has a second attribute called cost. This represents the relative expense (in time or other resources) of verifying the symptoms or performing the actions described at the node. Node strength and node cost are combined into a weighted entropy measure of useful information similar to that of [22].[3] This measure is used to identify the best child node when the user cannot answer the questions necessary to select a child with certainty.

The CHATKB inference algorithm identifies solution cases by searching through the top-level knowledge module first, and then through tool-level modules. The algorithm is outlined in Figure 2. Its input is an initial problem description and information about the context of the problem. Its output is the UPERIT template of the best matching case in the knowledge base.

When all UPERIT templates found are unsatisfactory and rejected by the user, node strengths must be revised. CHATKB uses a heuristic algorithm to recursively update the tool module leaf corresponding to

[3] Several other diagnosis and recovery expert systems use various entropy measures. Cost is not taken into account in FIS (Fault Isolation System) [19] or in GDE (General Diagnostic Engine [4]). Cost is considered in [2] but no combined entropy measure is used. An entropy measure taking into account cost is used in [14].

the rejected templates, all ancestors of this leaf, the top-level leaf corresponding to the active tool module, and all ancestors of this top-level leaf. After updating, case retrieval is attempted again. If no satisfactory cases are found after updating, the session log is saved as an unresolved case and the user is told to contact the CHAT support group.

- Step 1: Input an initial description of the design environment and problem symptoms.
- Step 2: Starting at the root of the top-level module, repeat the following until a leaf is reached:
  - Select a child of the current node which matches the inout environment attributes. If no child matches exactly, select the child with lowest entropy. If the list of children is exhausted, return to Step 1.
- Step 3: Activate the tool module corresponding to the top-level leaf reached.
- Step 4: Input a detailed description of the design tool(s) in use, tool options, technology, design task and subtask, actions taken, and messages received.
- Step 5: Starting at the root of the active tool module, repeat until a leaf is reached:
  - Select a child of the current node which matches the problem description given in Step 4. If no child matches exactly, select the child with lowest entropy. If the list of children is exhausted, backtrack to Step 2.
- Step 6: Retrieve all UPERIT template instances matching the index fields specified as critical at the tool module leaf reached. Rank the list of templates by collecting additional information from user. If no additional information is available, rank templates based on the number of index matches. Present the best matching template to the user.
- Step 7: If the user accepts the solution, done. Otherwise, if the template list is not exhausted, present the next best matching template to the user.

Figure 2: The CHATKB case retrieval algorithm.

It is noteworthy that CHATKB identifies cases as relevant using domain knowledge primarily. Only within a final subclass is similarity of case features taken into consideration. This is because deciding which features are genuinely predictive for complex case structures such as UPERIT templates is difficult. Automatically inferred predictiveness can be misleading when there are coincidental similarities among cases, which is especially likely with large cases or with a small set of cases [18].

6 The CHATKB system

We selected Knowledge Director as the main tool for implementing CHATKB. Knowledge Director is a classification and decision expert system shell which runs on OS/2 under Presentation Manager. Knowledge Director was chosen for two reasons. First, it is quite similar to Nexpert Object, a Neuron Data tool which we used in earlier work on SPICE. Second and more important, Knowledge Director is a highly flexible shell. It can accommodate declarative, hierarchical knowledge as found in the top-level and tool knowledge modules, and also the procedural knowledge associated with CHATKB front-end and back-end processing. In addition, Knowledge Director allows easy communication with external applications, which is important since UPERIT templates are stored in a relational database system.

As explained above, in CHATKB self-contained knowledge modules exist for the VLSI design domain as a whole and for each design tool. Separate procedural modules also exist for front-end and back-end initialization, knowledge maintenance, node strength updating, and entropy evaluation. The system currently has 14 tool modules. These contain 41 unique leaves and 242 internal nodes, with a depth of 4 to 17 levels from root to leaf. Leaves are typically shared among several branches, meaning that different branches converge to the same UPERIT templates. We tried to avoid structural duplication as much as possible in the tool modules, but the penalty for this was increased knowledge acquisition complexity, and also an increased number of parameters at each node, due to the need for taking into account more symptoms and conditions to uniquely identify the next node (decision step).

The use of navigation as opposed to search as the major mode of inference in CHATKB permits the CHATKB user interface to be conversational. To start a consultation session CHATKB asks the user to describe the design and design environment by selecting options from list boxes describing platforms, design systems, methodologies, technologies, and problem types. Manual data entry is minimized by using pop-up menus, and online explanations are available for each menu. As the session progresses, CHATKB questions are customized for the error episode being analyzed: only applicable answers and choices are shown to the user.

Several features of the CHATKB user interface were influenced by lessons learned from episodes of miscommunication evident in the free-format CHAT case transcripts. In particular, we saw the need to provide

CHATKB with a natural language phrase search and match capability. Words and phrases can be matched based on word-roots and tables of equivalent phrases, such as *very, much, a lot, lots of* and *VLSI, IC, LSI.* This capability saves users from remembering the exact wording of messages.

7 Experimental evaluation

The effectiveness of CHATKB has been evaluated experimentally through offline simulation and interactive simulation.

In the offline simulation experiment, we ran the 399 initial cases extracted from CHAT through CHATKB in batch mode and recorded coverage (how many episodes were diagnosed correctly), performance (how much search and dialogue were reduced), and originality (how many new or different recovery initiatives were suggested). We did not attempt to measure usability improvements or time savings, so a solution recorded in CHAT as reached after many rounds of email was evaluated as equal to a solution reached by CHATKB if it suggested the same recovery plan.

The diagnoses and plans produced by CHATKB were compared to the problem solutions recorded in CHAT manually. CHATKB generated 1365 diagnoses for the initial 399 cases. The reason for so many diagnoses is that in its offline mode CHATKB investigates all potential alternatives rather than choosing one alternative by asking the user for more information. For example, when the description of the problem does not specify whether the technology in use is CMOS or BiCMOS, both possibilities are investigated.

Of the 1365 diagnoses generated, 764 are *unknown*, *not supported*, *not applicable*, or *incomplete*. The remaining diagnoses included 374 distinct diagnoses, each accompanied by a recommended recovery plan. These plans include 15 solutions for problems which had not been resolved in the CHAT database yet. (Over 10% of the initial 399 problems had not been resolved at the time the database was backed up.) In more detail:

- 318 plans are identical with plans in CHAT;
- 29 plans reach the same conclusion with 1 or more extra steps;
- 12 plans reach the same conclusion with 1 or more fewer steps;
- 15 plans are new, built from a combination of existing plans and subplans. Of these 15 plans, 3 are valid solutions, 7 are invalid, and 5 are too complex to evaluate.

In the second experiment, we compared human subjects performing the same task using CHATKB and using CHAT. Unfortunately we could only run experimental sessions with eight subjects, so this experiment should be viewed as a trial study.

The eight subjects who took part were owners of 47 of the initial 399 problems. Each went through a standard online CHATKB introductory session. Then in ten sessions, each person solved ten problems using CHATKB. In each of these sessions, the subject first reviewed a hard copy of all messages and files in CHAT for one problem, without seeing any diagnosis or recovery plan recorded in CHAT. The subject then reported the problem to CHATKB. All interaction with CHATKB was logged automatically; timings and verbal comments were recorded manually. Over the 80 sessions, the time to solve a problem ranged from 14 to 28 minutes, with an average of 17 minutes. Users could spend as much time as they wished exploring the system; on average they spent three minutes per session. In 35% of sessions the user was very satisfied, in 40% satisfied, in 15% partially satisfied, and in 10% not satisfied or undecided.

8 Conclusion

Statistics from the CHAT support group show that the number of new problems reported to CHAT has decreased, even though usage of the system has steadily increased. Nevertheless, the average time from reporting a problem until its resolution has not decreased. In conjunction, these two facts are a window of opportunity for CHATKB. A stable knowledge base can contain solutions to most user problems, and these solutions can be delivered much faster than with CHAT. A common problem with expert systems (especially medical ones [7]) is that users do not like using them, so their benefits are not realized. The experiment reported in the previous section indicates that on the whole, users like CHATKB, so we are confident the system will be used.

At first glance it may appear that the knowledge in CHATKB is shallow, since most knowledge is in the form of cases and no elaborate reasoning algorithm is used. The opposite is in fact true. CHATKB can index onto appropriate cases with almost no search because it possesses general knowledge about the VLSI design domain. Without this deep knowledge, it would not be possible to provide interactive, cooperative, and intelligent consultation and dialogue with users. The same deep knowledge also makes it possible to add knowledge about new cases and tools easily to CHATKB.

The architecture of CHATKB combines problem-solving by classification and by case retrieval. Other systems for fault diagnosis and repair have used somewhat similar hierarchical organizations of knowledge, with inference using domain knowledge first, then similarity between cases [15, 14]. The categorization-based architecture appears to be well-suited for diagnostic problem-solving in general, and we have also used it in other expert systems [9]. Interestingly, it is very similar to an architecture proposed recently as a model of human problem-solving [6].

The deepest knowledge in CHATKB is the classification scheme for episodes of error interpretation and recovery that is part of our framework for understanding such episodes. Like some previous theories of human information-processing [8], this framework treats systematic errors and correct performance as two sides of the same coin. It extends previous theories to accommodate environment errors also, and most important, human and system initiatives to recover from errors. We believe the framework provides a well-defined, robust methodology for capturing and representing the expertise required to use VLSI design tools.

Acknowledgements. We are grateful for assistance from the CHAT support group at IBM, and for grants from the Powell Foundation and the National Science Foundation (No. IRI-9110813) at UCSD.

References

[1] T. Cain, M. Pazzani, and G. Silverstein. Using domain knowledge to influence similarity judgement. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 191–199. Morgan Kaufmann Publishers, Inc., May 1991.

[2] R. R. Cantone, W. B. Lander, M. P. Marrone, and M. W. Gaynor. IN-ATE/e: interpreting high-level fault modes. In *Proceedings of the First IEEE Conference on Artificial Intelligence for Applications (CAIA)*, pages 470–475, December 1984.

[3] W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[4] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[5] Richard O. Duda and Peter E. Hart. *Pattern classification and scene analysis*. John Wiley & Sons, Inc., 1973.

[6] Douglas H. Fisher and Jungsoon Yoo. Categorization, concept learning, and problem-solving: A unifying view. In *Categorization by Humans and Machines*. Academic Press, 1992.

[7] D. E. Forsythe, B. G. Buchanan, J. A. Osheroff, and R. A. Miller. Expanding the concept of medical information: an observational study of physicians' information needs. *Computers and Biomedical Research*, 25(2):181–200, April 1992.

[8] L.P. Goodstein, H.B. Andersen, and S.E. Olsen, editors. *Tasks, errors, and mental models: A festschrift to celebrate the 60th birthday of Professor Jens Rasmussen*. Taylor & Francis, 1988.

[9] Amir Hekmatpour and Charles Elkan. A multimedia expert system for wafer polisher maintenance (abstract). In *Proceedings of the IEEE International Conference on Artificial Intelligence for Applications*, March 1993.

[10] Amir Hekmatpour, Alex Orailoglu, and Paul Chau. Hierarchical modeling of the VLSI design process. *IEEE Expert*, 6(2):56–70, April 1991.

[11] Janet L. Kolodner. *Retrieval and organizational strategies in conceptual memory: A computer model*. Lawrence Erlbaum Associates, 1984.

[12] Phyllis Koton. Reasoning about evidence in causal explanation. In *Proceedings of the National Conference on Artificial Intelligence*, 1988.

[13] M. Lebowitz. Concept learning in a rich input domain: Generalization-based memory. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann Publishers, Inc., 1986.

[14] W. Y. Lee, S. M. Alexander, and J. H. Graham. A diagnostic expert system prototype for CIM. *Computers and Industrial Engineering*, 22(3):337–352, July 1992.

[15] C. A. Marsh. The ISA expert system: a prototype system for failure diagnosis on the space station. In *Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-88)*, pages 60–74, June 1988.

[16] Donald A. Norman. Position paper on human error. In *NATO Advanced Research Workshop on Human Error*, Italy, 1983.

[17] Donald A. Norman and Tim Shallice. Attention to action: Willed and automated control of behavior. Technical Report CHIP99, Center for Human Information Processing, University of California, San Diego, 1980.

[18] Michael Pazzani. *Creating a memory of causal relationships: An integration of empirical and explanation-based learning methods*. Lawrence Erlbaum Associates, 1990.

[19] F. Pipitone. The *Fault Isolation System* electronic troubleshooting system. *IEEE Computer*, 19:68–76, 1985.

[20] J. Rasmussen. Human errors: a taxonomy for describing human malfunction in industrial installations. *Journal of Occupational Accidents*, 4:311–335, 1982.

[21] J. Reason. A framework for classifying errors. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology and Human Error*. John Wiley & Sons, Inc., 1987.

[22] B. D. Sharma, J. Mitter, and M. Mohan. On measures of 'useful' information. *Information and Control*, 39:323–336, 1987.

[23] E. Simoudis and J. S. Miller. The application of CBR to help desk applications. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 25–36. Morgan Kaufmann Publishers, Inc., May 1991.

[24] Stephen Slade. Case-based reasoning: A research paradigm. *AI Magazine*, 12(1):42–55, 1991.