Eliciting knowledge from experts is decidedly difficult. By restricting the forms of required knowledge without sacrificing system performance, developers of CRIB have designed a successful diagnostic tool.

SPECIAL FEATURE

CRIB: Computer Fault-finding Through Knowledge Engineering

CRIB, or computer retrieval incidence bank, is one of a small number of expert systems [1,2] designed for computer fault diagnosis. Although it possesses little of the structural or functional knowledge common in more advanced expert systems, this system is a highly flexible, userfriendly, and pattern-directed inference system that is adequate for both hardware and software fault diagnosis.

In this article, CRIB is described in terms of its generation, or phases of development, to emphasize the importance of predevelopment and post-development system analysis in building a viable expert system. Three systems development phases (not dissimilar to the development phases of any commercial computer system) have been described:

(1) The elicitation, or selection, of data to form a knowledge base consisting of basic symptoms in group and/or fault pairings.
(2) The design, development, and testing of the expert system.
(3) The continuous validation, fine-tuning, and im provement of the expert system.

The before and after of an expert system

Prior to describing how CRIB's knowledge base was developed, it is helpful to make an analogical distinction between the characteristics of an expert system and the characteristics of a standard database system. In our analogy, human *experts will* be compared to human practitioners. *

* Practioners, in this context, are those individuals who use some formal system of knowledge to perform their everyday tasks; the experts are those who define what the system is and how it should be used.

Elicitation: from experts or practitioners? "Expertise" is a relative term; an expert is only an expert relative to the practitioners in the field. Moreover, experts are always a tiny minority of the whole group. However, the question arises, "Can a large group of practitioners, working together, produce better results than an expert?" If so, an expert system based on the aggregate practitioners' knowledge can potentially outperform one based on the knowledge of a single expert. Yet, the answer to the question must surely depend to some extent on the sort of knowledge involved.

Knowledge that is fragmentary, consisting largely of facts that can be added together without inconsistency, could be used to produce a viable knowledge base for an expert system. Indeed, practitioners acting separately (or at least only loosely connected) and having different experiences and therefore different knowledge could provide a larger collective base of knowledge than the expert. The expert has only his own experience to rely on and, though he might have a far wider experience than any one prac- his knowledge cannot match the aggregated knowledge of an entire group of practioners. If, on the other hand, the system requires a structural or systemic knowledge base in which facts are connected by inferences of various sorts, then the knowledge of the single expert would be better. In fact, the expertise of individuals often stems from their ability to see the entire structure of a domain and not merely the contents.

Further discussion on knowledge-based artificial intelligence[3] and theories of knowledge[4] can be found in other sources. The only point we must understand here is that the sort of knowledge one must have to be effective in a particular domain depends on the sorts of activity within that domain. For example, the knowledge required to be a

0018-9162/84/0300-0076$01.00 © 1984 IEEE

good bricklayer is mainly additive. The bricklayer's knowledge must consist of facts such as how to chop a brick in half, what proportions of sand and cement make good mortar, and how to build a corner. On the other hand, the knowledge required by the contractor employing the bricklayer is structural (or inferential) in the main. The contractor must know how to time the work of subcontractors (especially when the work of one depends on another finishing); how to balance incoming and outgoing payments so as not to go bankrupt; when to quote a high price for a job he cannot find time to do.

The type of knowledge required for completing a task therefore appears to be dependent on the domain. All the early expert systems [5] were developed for academic or scientific areas of applications. These systems required an inferential type of knowledge for finding and making explicit that which was previously implicit and therefore clearly fit into the "expert elicitation" category. True, they also used a large body of factual knowledge, but this knowledge was decidedly subordinate to the inferential knowledge that reflected the "expertise" of the systems. The facts were taken for granted and formed a significant but largely static part of the system.

Consequently, there should be areas of human activity in which the notion of an expert is not inappropriate and in which the expertise of such an individual is primarily dependent on a large and ever-changing body of facts. The expert in such a field will have the structuring skills of the builder in our preceding example, but the aggregated practitioners will have more factual knowledge, which will be the source of the initial knowledge base. Computer fault diagnosis is certainly one of many applications requiring this type of knowledge-base. *

Eliciting data for the expert system. Since most knowledge-based systems maintain the distinction between static knowledge, or facts, and methods of inference, heuristics can aid in the initial process of building a knowledge base for an expert system. We can also organize techniques for eliciting data around these categories. When working with the human expert, both types of knowledge can be readily integrated; however, working with a collection of people is different primarily because facts are only additive if inconsistency and contradiction are allowable. An expert system cannot afford to appear inconsistent and contradictory to the user; for this reason, some sort of semantic network is usually incorporated to stop potential inconsistencies by providing links between the facts, though all of the problems of building networks with.sufficient expressive power have not yet been solved. [6] However, a theoretical input on behalf of the designer can also overcome this problem of multiple sources with possibly conflicting knowledge.

In CRIB, the stored knowledge is reduced to simple facts without reducing the system's efficiency as a diagnostic aid. These simple facts (basically the symptom group/fault pairings discussed later in "Forms of knowledge for computer diagnosis" on p. 78) are easy to gather using a variety of techniques, making a semantic network unnecessary.

Methods of inference in expert systems. Most expert systems using production system methodology necessarily force all inference procedures into the uniform situation/action pair. The elicitation is therefore characterized by a set of rules rather than by a set of inferences. Collecting methods of inference from an aggregate of practitioners involves a second level of system design, since some form of induction must be used to capture commonalities among the members of the group. Alternatively, systems designers could supply methods of inference that are based on theories of what can be deduced from combinations of data items in a specific domain.

In CRIB, stored knowledge is reduced to simple facts without reducing the system's efficiency as a diagnostic aid.

A study prior to CRIB, project Deeman,* suggested that a hierarchical model of the broken machine be adopted to guide a step-by-step search for faulty firmware, hardware, or software. The designer supplied a model of competence, which the system then used as the universal method of making inferences and guiding the user. Johnson 7 makes the point that most artificial intelligence systems are in fact models of competence in such areas as problem-solving, natural language, and pattern recognition.

Validation and improvement. If an expert system is indeed a model of competence, then we must be prepared to change the model to suit the experience of the users. The same problem occurs after the system is developed as before elicitation; because practitioners mainly use the system, it is their experience that is useful in validating the model. But can this experience also be used to improve the model? In general, system users come to two different conclusions after gaining experience with the system: the model is either (1) inadequate as a model of competence (that is, the system is "incompetent"), or (2) it is competent but is failing to cope with a wide enough range of new situations and methods of working. This leads to a distinction between validation and improvement. An expert system that is improving in the more fundamental sense must be gaining new knowledge; however, validation can be carried out by restructuring existing knowledge. The same distinction also applies when considering evolutionary changes to a system. Continuous validation through day-to-day use by practitioners can affect small inadequacies of a system, like the tuning of a radio for better reception, but new knowledge is usually introduced by an expert, who, in this analogy, would be the one to select a "better" radio station.

*Almost any skilled trade would utilize an initial knowledge base built from an aggregate of facts, as well as such diverse fields as navigation, production scheduling, or information services.

* "Deemen" is derived from the original term, D-men used as a reference to diagnostic men.

Another consideration is whether system improvement can come from an aggregate of practitioners in the same way that elicitation can. Practitioners, by the nature of their job, do not normally generate new knowledge. If they do, it is more likely to be factual than inferential-the sort of knowledge that an expert system should be able to handle by the "fine-tuning" process of continuous validation.

CRIB: a computer engineer's diagnostic aid

The initial aims for CRIB were strictly commercial: to reduce the cost of training new engineers and existing engineers on new equipment and to increase productivity by reducing the average time per fault investigation. With constraints like these, a highly flexible, user-friendly system was clearly needed. Early decisions included the choosing of (1) an object computer (the International Computers Limited, or ICL, 1903a) and language (ICL's Coral 66), (2) a "patient" computer for a feasibility study (the ICL minicomputer 2903), and (3) a set of multiuser software standards (the ICL Driver). Ultimately, it was decided to make CRIB compatible with the content addressable file store, or CAFS, database processor because of its wealth of associated software packages and analysis techniques and its high efficiency as a pattern-directed database system. * Using the existing logical structure, the operational speed of the CAFS version was 20 to 30 times greater than all previously tested database processors.

The whole CRIB system consists of four programs (Figure 1). The main program carries out the executive function and is called *Diagnose.* It interfaces with the user (a field engineer) through a simple jargon-English translation package. This translator allows the engineer to communicate with CRIB in a fairly natural way to describe symptoms in the form of actions carried out and control housekeeping functions (such as the investigation log). Some examples and explanations of the resulting terminal messages[2] are

LOGIN FRED, HERE (start investigation)
PROGRAM LOAD FROM DISC NOT OK (symptom description)
PROGRAM LOADED FROM CR NOT OK (symptom description)
ALTER AND DISPLAY ONES OK (symptom description)
CHECK HAND SWITCHES INDIRECT (action description)
LOGOUT OK (terminate investigation)

*Early versions of CAFS used standard file techniques for database management.

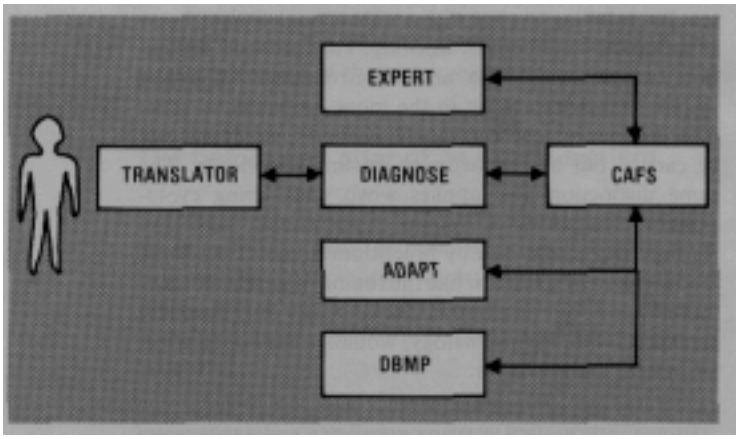


Figure 1. CRIB program structure.

Some degree of self-improvement and fine tuning is done by *Adapt,* a program that examines the logs of previous investigations and updates times of actions performed and investigation times after a complete match with a symptom group. It also handles subgroups.

Major restructuring of the database and the insertion of new knowledge are done through *Expert.* This specialized access to the database is performed with the help of the system designer.

DBMP, or the database management program, is the fourth,

Forms of knowledge for computer diagnosis. Two questions posed at the beginning of system design were: (1) what sort of knowledge does an engineer need to find faults on malfunctioning computers, and (2) how does the engineer use this knowledge to find and correct the fault? The Deemen project made it evident to all of us on the research team that the expertise of most field engineers is not in assuring correct electronic functioning of the machine (let alone in digital circuits pathology), but rather in module interfaces. Therefore, the approach to diagnosis through simulation of machine function was quickly rejected. Most previous attempts along these lines failed through the enormity of the task; a computer is just too complex to be simulated at any level useful for diagnosis (although recently a more plausible approach has been taken ı). Instead, we concentrated on the notion of fieldreplaceable parts, or subunits, and the interfaces between them. This focus allowed us to drop the concept of "a fault" in that it specifically implied a particular malfunction of a particular component. All that was really needed was the location of the fault within a subunit. Consequently, the only knowledge needed about the structure of the machine was the hierarchy of subunits that expressed the interrelationships. Where subunits shared hardware interfaces (such as a cable or connector) relationships between them were represented as additional subunits at their same level (Figure 2).[8]

Although there are good reasons for taking this approach, more justification comes about through the methodology suggested in the Deemen reports; an easy-to-follow routine for diagnosis was suggested in an attempt to give engineers a fail-safe method of proceeding when all else fails. This procedure is called TOAST and consists of the following fault-diagnosis procedures:

Test: Carry out an appropriate test on the machine.
Observe: Observe and record the results.
Analyze: Analyze test results and determine whether to split or test.
Split: Split the faulty subsystem into faulty and nonfaulty parts.
Test: Generate an appropriate test for the faulty subsystem.

This testing spiral takes the engineer down the hierarchy of subunits until the location of the fault is sufficiently pinpointed to repair or replace a module.

TOAST was not envisaged as operating on a fixed hierarchy of subunits as CRIB does, but it did form the basis of CRIB as a model of competence in computer diagnosis. For TOAST to be operational we needed to determine the knowledge required for the Analyze phase. We assumed that the engineers had to know the machine functions, and we consequently encountered a problem in keeping the knowledge forms as simple as possible. The solution to the problem was to represent the analysis as a simple association of symptoms and faulty subunits. This associational process raised two questions: (1) How do we know which pairs to put in the database, and (2) Do symptoms uniquely identify faulty subunits? The first question is concerned with elicitation and will be discussed later in this article. In general, there is no way to discover the right pairing analytically. The second question, however, is clearly something only an expert can answer.

The experts we worked with told us that symptoms do not uniquely identify faulty subunits; some symptoms are common to many faults and therefore require more information. At this point, we were forced to break with the TOAST method; wanting to include knowledge of machine function, we added information to the pair that included all other symptoms observed prior to the current one. This addition increased our chances of uniquely identifying a faulty subunit, since we could observe 10 or 15 symptoms during an investigation.

The question therefore takes on another dimension: Does a *group* of symptoms uniquely identify a faulty subunit? Apparently, the answer to this is "yes"-as long as they are the right symptoms. In other words, if an engineer simply observes a group of symptoms and finds the fault, then the engineer's analytic skills are reflected in the choice of actions taken to yield the symptoms in the group. Consequently, we do not need to represent the analytic skills explicitly, since they are represented implicitly through the symptom groupings.

Faults may exist that cannot be characterized by any group of symptoms. Though we did not discover any faults of this nature, these are extremely difficult for an engineer to diagnose and virtually impossible for CRIB.

This solution is fine, except that symptoms are not simultaneously apparent. Some are observed only as a result of lengthy and intricate operations. To simplify observations, we need to know which symptoms are the most important. This requirement takes us to the last part of the TOAST cycle: generating a test. Since CRIB has no functional knowledge, its assessment must be based on attributes or actions that do (or do not, as the case may be) yield symptoms in the various groupings. Attributes, uch as the time it takes to perform an action; how much information it would yield if it were observed (in terms of the number of groups in which it occurs); and how long the investigation might take after it has been observed, are only heuristically adequate in this case. However, since the final aim is to find all symptoms in any one group that might guarantee success, the heuristic element is concerned only with means and not ends.

Knowledge categories in CRIB. The knowledge needed for computer diagnosis in CRIB is summarized in this way:

- *K1* = A hierarchy of subunits representing the structure of the machine as a collection of repairable or replaceable parts.
- *K2* = A number of symptom group/subunit pairings that represent successful fault investigations.
- *K3* = Attributes of symptom-related actions that are heuristically adequate to determine the next action.
- *K4* = Procedures for operating a modified TOAST cycle in order to progress down the hierarchy to the faulty subunit.

These knowledge categories form a model of competence in computer fault diagnosis. This model is based on the acknowledged competence of experts and practitioners and not on analysis of machine function or theories in either computer design or construction.

Elicitation or knowledge in CRIB. The sort of discussion presented earlier in "Elicitation: from experts or practitioners?" came several years after the derivation of the first-stage database for CRIB. However, a short historical summary is appropriate here, since we can now criticize it in the light of new knowledge. The categories of knowledge (*K1*, *K2*, *K3*, and *K4*) were elicited from various sources; knowledge category *K4* was based on the TOAST cycle.

The hierarchy of subunits, or knowledge category K1, was formulated in our discussions with field engineering trainers. The number of levels and the choice of names are

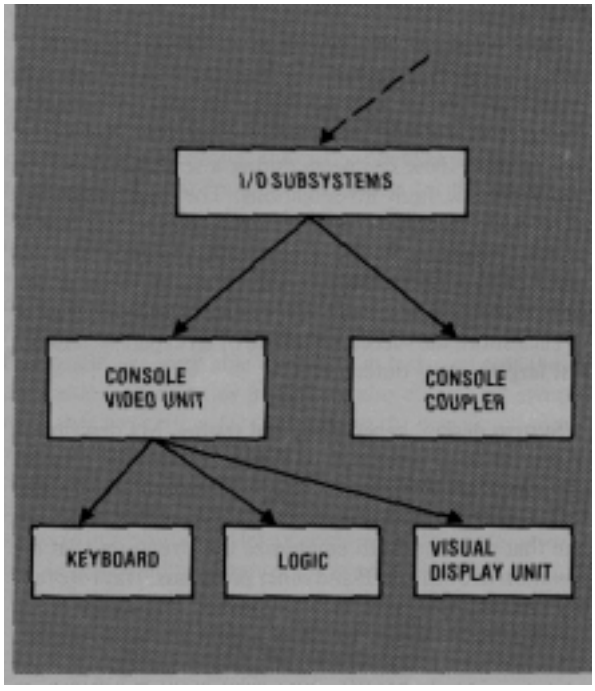


Figure 2. Part of a hierarchical model. [8]

fairly arbitrary, but *K1* does provide a comprehensive breakdown* of the ICL 2903 to the level of field-replaceable units, which was its primary goal. A small part of *K1* is illustrated in Figure 2.

The bulk of the knowledge category K2 was derived from a set of fault-finding guides for the ICL 2903. These guides had been produced by an engineer who applied his expertise in diagnostic fault-finding to the then-new machine. Though the guides were to a large extent inadequate, they did provide a first-stage database that hopefully could be improved. Our interpretation of these guides could therefore be called "elicitation from an expert," except that the guides needed a large amount of interpretation to fit the CRIB symptom group/ subunit pairings. in particular, the guides had a flexible notion of hierarchy and did not adhere to a strict TOAST methodology. Further elicitation came from our interpretation of tape recordings produced by engineers in the course of actual fault diagnosis in the field. The engineer's monologue was timed to approximate how long his actions (such as the carrying out of tests) took. Since the fault-finding guides had no action timings, the tapes gave us very valuable information for knowledge category K3. We

The first stage of elicitation-determining where specific knowledge should be derived from-is critical to the success of an expert system.

could also determine which symptom groups led the engineer to the fault; this helped to add new knowledge to the database. Though the exercise was unfortunately cut short, we ultimately could have validated all the faultfinding guide data.

With hindsight, our first-stage elicitation should have come directly from an expert, since there were no practitioners at the time with any direct experience. Although the tapes were useful, more information could have been gained from these engineers during a series of interviews about specific fault investigations. The data could then have been translated into a form suitable for CRIB without engaging in the sort of interpretation-at-a-distance demanded by the protocol-recording method. We cannot overemphasize the importance of this initial stage of elicitation; the success or failure of an expert system can rest largely on its outcome.

System design, generation, and testing. My description of the internal workings of CRIB follows the analysis given by Hays-Roth et al. [9] Other, perhaps more straightforward accounts may be found elsewhere. [8,10] Accounts like that of Hays-Roth emphasize the structures that are common to both CRIB and other programs. Hays-Roth et al., for example, give a taxonomy of knowledge-based systems and attempt to fit a published work into its framework. Our categories of knowledge have many of the characteristics of what they call a pattern-directed inference system, or PDIS. The group/subunit pairings can be seen as expressing this inference:

If all these systems have been observed
then assume that the fault is in this subunit.

Such an antecedent-consequent pair is usually called a *rule.* The procedures in *K4* clearly have a matching function, in part; as in TOAST, the process of looking for a suitable split to be made involves matching a current observed-symptom set with each of the pairings in the database. The symptom groups are therefore *patterns* and the pairings are *inferences.*

One of the main features of a PDIS is the separation of data and control. in general, a PDIS consists of a large, constantly changing body of data and a relatively small, fixed set of procedures. These procedures form the exec*utive* of the PDIS. Hays-Roth et al. discuss PDIS executive in four parts: selection, matching, scheduling, and execution. Not all such systems have all four parts, but we can still discuss CRIB in terms of selection, matching, and scheduling.

(1) Selection. With very large databases, we would like to filter the data before executing the expensive matching process. The selection process in CRIB gives the user extra control over the system. CRIB can be used more or less as an inquiry database if so directed, and can therefore quickly inform the engineer whether he has located the fault correctly or not. The user is able to direct the program to look only for matches among symptom groups paired with any chosen subunit and not, as is normally the case, for matches among symptom groups paired with all subunits below the one currently assumed to be faulty. This capability can improve search efficiency and is especially useful to the engineer during the process of eliminating possible problem sources.

(2) Matching. The major function of an *executive is* to match observed symptoms, input by the user, and symptom groups in the database. This matching is performed by the CAFS database processor. Inquiries in the form of Boolean expressions of attributes known to the database are entered into CAFS processor in the form of a microcode program, The CAFS then returns selected parts from all records that satisfy the request to the mainframe by way of a direct memory access, or DMA, link. No complicated matching algorithms are needed, since CAFS operates independently of the calling program. Information about the CAFS side of the CRIB implementation, including the relational analysis that preceded the setting up of the database, is discussed by Addis.8 Technical information about CAFS can be found in work by Babb. [11]

(3) Scheduling. There are three possible results to matching. The first is that there is only one match with a rule in the database; in this case, scheduling is not needed. The other two are more difficult; either there is more than one match (requiring conflict resolution) or, as is usually the case in CRIB, there are no complete matches, only partial ones. Scheduling is the process of deciding what to do next; meta-rules, which describe how to proceed in situations of imperfect matching, are written to aid in this decision-making process. In CRIB, the meta-rules for

* It serves that any breakdown is satisfactory as long as it does not conflict with the structure of the physical system as a collection of parts.

handling partial matches are aimed at suggesting actions to the engineer that will yield symptoms to complete a partial match.* The meta-rules are

(1) Choose the only remaining symptom in a group; all others should have been observed already.
(2) Choose the symptom with the highest membership in a partially matched group; This symptom, if it were observed, would give the engineer the most information.
(3) Resolve conflicts in meta-rules I and 2 by taking the symptom with the shortest time factor, (the sum of the time it takes to determine whether the symptom is present or not and the subsequent (postobservation) time that it takes to investigate the symptom).

In many situations, especially early in an investigation, even the preceding rules do not yield a unique choice; in such situations the rules are applied five times to eliminate impossible symptoms, and five symptoms are chosen. This procedure affords the engineer an element of choice in his actions in that the meta-rules are only heuristically adequate, as previously mentioned, since the assessment is not based on a machine function. CRIB has no explicit knowledge about maintaining a line of investigation or of functional connections between symptoms.

Subgroups. Although TOAST represents a good model of diagnostic competence, engineers do not always like to use such models. Some engineers are skilled at recognizing the emergence of symptom patterns before all the faultconfirming symptoms have been observed. This heuristic knowledge can be captured in a set of rules of the following form:

If all of these symptoms are present
Then assume that others that have not yet been observed are present also.

A set of symptoms that implies a faulty subunit is a subset of the whole set of symptoms. In CRIB terminology, the confirming set is called a *keygroup* and the subset is a *subgroup*. The ideal is to match a whole keygroup; however, a match with a subgroup is considered heuristically adequate to make progress in the investigation. Since subgroups often reflect the individual engineer's method of working more than the keygroups, they must be confirmed by several engineers before being accepted into a system. The Adapt program checks for the confirmation of a subgroup that is used successfully and rejects those subgroups that are disconfirmed by failure. More important, Adapt discovers new subgroups by applying the following meta-rule when analyzing the investigation log:

If action suggestions resulting from a partial match result in the completion of that match
Then insert the symptoms making up the partial match into the database as a possible subgroup.

Implementation of rules and meta-rules. The four previously discussed meta-rules are implemented conventionally through fixed procedures. Since they form the analytic part of the model of competence, there is no need to represent them explicitly. However, rules I and 2, when instantiated by particular symptom groupings, need a flexible representation to allow for additions and modifications to these rules as the system grows and changes. The CAFS architecture fortunately allows the programmer to think in terms of *relations* among the data contained on the device, this facilitates the implementation of an if-then rule, and all rules in the CRIB database are consequently represented as relations among symptom groups and subunits (as in the case of keygroups in rule 1), or other groups (as in the case of subgroups in rule 2). Again, there is no need for explicit rule "languages" with correspondingly complicated matching algorithms; the efficiency of the CAFS hardware makes them redundant.

Validation, improvement, and performance. The phase immediately following system generation was as difficult to execute as eliciting the database from a variety of sources. My colleagues and I still had relatively little experience, either directly or indirectly, of real faults on the chosen machine. However, we could validate CRIB's competence, if we assumed that the correspondence between symptoms and the actions designed to reveal them

If all went well, CRIB would suggest the repair of a terminal subunit. This was indeed the outcome in a test involving 200 engineers.

was accurate and that the fault-finding guide data was not too far from the truth. A relatively new user (an engineer new to the ICL 2903) was asked to be completely guided by CRIB and "carry out" the actions suggested by the system. If all went well, we would have expected CRIB to suggest as a final action the replacement or repair of a terminal subunit. This was the case in a large number of "investigations" carried out by almost 200 engineers. However, in some cases, especially where the chain of suggested actions was long, the really relevant actions were not displayed soon enough. The reason for this problem had to do with scheduling of actions after a partial match (as suggested in meta-rule 2). Engineers would not make such obvious errors because they rely on the functional dependence between symptoms; CRIB cannot use this fact, but with the filtering facility previously described in this article, we were able to resolve at least some of these difficulties. During this phase, we also eliminated symptoms that were either (1) irrelevant to the new situation of working with a computer aid, or (2) unhelpful to diagnosis. After several weeks of testing in this manner and incorporating data captured from taped engineer protocols, we were in a position to let an engineer (in this case, an expert in fault-finding diagnostics) use CRIB to find "real" faults. [8] Although the engineer had some difficulty, (particularly with the problems of unambiguous communication of symptoms to the system), he found CRIB's diagnostic methodology viable and considered the system a competent diagnostician.

*This technique is similar to backward chaining in Mycin and other similar systems in which assumptions are made and information to confirm or deny the assumptions is requested.

Assessment of performance. CRIB can certainly help an engineer find faults on malfunctioning computers when these faults have been successfuly remedied before. The majority of the day-to-day work of a field engineer involves diagnosing recurring problems. If engineers do not spot a fault as a known one (whether or not they are using a system such as CRIB), they must fall back on basic knowledge of both the particular machine they are working on (possibly functional knowledge) and diagnosis procedures in general. Deemen was a project concerned with providing a general methodology in such circumstances.

What can be said about CRIB when it is faced with an unknown fault? Certainly the correct symptom group is not in the database. If it were, then the fault would be known. It is possible, but not likely, that the required group is a subset of one that is present (perhaps even identical). More probable is that (1) the group involves symptoms that have not been observed before, or (2) completely new actions would have to be carried out to uncover the fault. In these cases, CRIB, with its complete lack of explicit functional knowledge, may not be of great help. However, because the correct group contains many symptoms that are known and present in other groups, and because these groups are likely to be associated with the subunit exhibiting the unknown fault, the engineer could be led to the correct faulty subunit by following TOAST down the hierarchy. This process has been observed when using CRIB in an unstructured way (for example, ignoring suggestions and putting in unsolicited symptoms). It is difficult to see how any expert system can be made to function effectively in such a situation; a doctor faced with a new strain of virus in a sick patient is in similar difficulty until an expert microbiologist comes to the rescue. Further information on and a proposed solution to the "new faults" problem are given by Addis.[10]

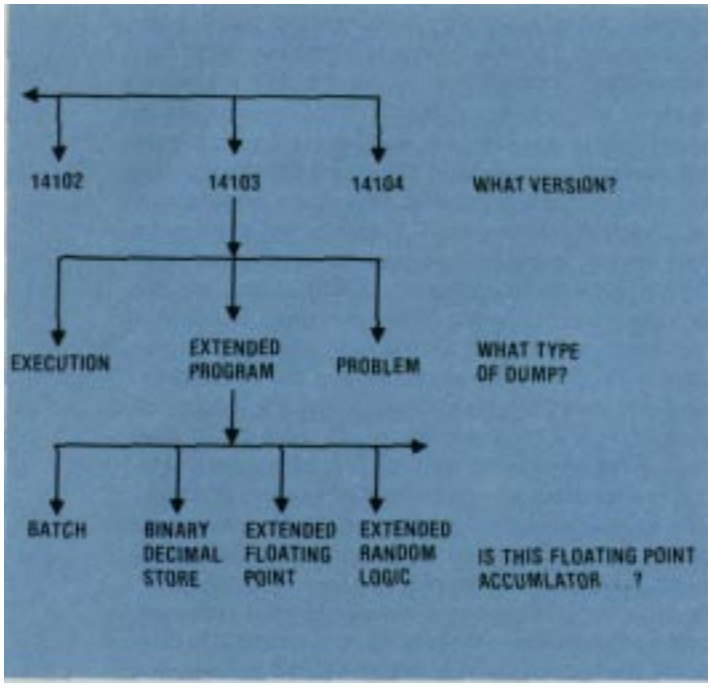


Figure 3. Part of the VM E/K structure. [10]

Soft CRIB: an exercise in generalization

The structures and mechanisms of the CRIB system can be described independently from the content of the data that drives it; consequently, as a production system it can be applied to more than just computer diagnosis.*

The most obvious application for CRIB was the diagnosis of software faults; this diagnosis was similar to the diagnosis of hardware faults. Within ICL, dump analyzers had already gathered data on faults exhibited by a new large-scale operating system called the virtual machine environment (version K), or VME/K. Most of the concepts in CRIB carried across to the software domain, albeit with reinterpretation. However, a major disparity was that large operating systems such as VME/K were not built hierarchically.

Although modular concepts are often incorprated, the notion of a repairable module does not exist for software, software modules do not fail in the same way that a malfunctioning component of a hardware module fails. Rather, software components contain bugs, which are essentially design errors. Fixing the bug in a particular module often influences several other modules, but bringing a broken component up to specification does not.

Consequently, because no general validity for a structural hierarchy of software modules exists, my colleagues and I were obliged to create a valid hierarchical model to circumvent the loss of this valuable asset within CRIB, which we called Soft CRIB. The solution to our problem was a hierarchy of attributes that describe the different levels of detail within the operating system. Suitable attributes were chosen to partition bugs and form a paradigmatic representation of the operating system from a diagnostic point of view. Attributes immediately subordinate to a common attribute can then be used to produce several answers to a single question (see Figure 3). [10]

Having solved this problem, the remaining correspondences were straightforward. Symptoms remained as symptoms, and key groups of symptoms corresponded to a *trace* (the set of symptoms recorded by dump experts performing debugging operations). Actions were classified as either *patches* (that is, pieces of code inserted to fix a bug) or *questions* that, when answered by the user, yielded further symptoms of that particular bug. Instead of establishing target performance times, each action was given a priority ranking to make sure that the system immediately asked important questions. Again, since times were irrelevant to this exercise, the subsequent investigation time was reinterpreted as an additive priority modifier; this reinterpretation ensured the immediate issuance of questions associated with attributes higher up the hierarchy.

Other features of SoftCRIB concerning interaction with and use of the new system have been presented by Addis, [10] but this article has summarized the most important parts of the transformation.

*CRIB is similar, in respect, to Mycin, a system that originated at Stanford University and was named after a series of drugs suffixed by mycin. The generation of Puff,5 Prospector, and Guidon 13 from the Mycin or Emycin (Essential Mycin)14 system architecture is proof of Mycin's wide applicability.

It has been said that an expert system should be able to
explain itself to the user just as a human expert can. Claims have also been made for "natural language" interfacing; creating interactive systems with mixed initiative; having incremental databases; and being adaptive and heuristic in operation. All these attributes are highly desirable and even necessary, but they are reduced to incidental window dressing unless the system actually possesses expertise in itsfield. Elicitation of expertise is crucial to the initial success of the system, and adaptability and enhanceability are crucial to its continued success. It is of little use to attempt a full-scale representation of expertise in the elicitation phase if the resulting system is difficult to modify and to improve. It is equally futile to program elaborate schemes for fine-tuning a database containing information that does not properly represent the desired expertise. The system should capture only the area of expertise that is sure to be useful to all practitioners, and that can be readily modified through straightforward improvement schemes. CRIB partly meets these demands but is now seen more as a prototype for better systems. •

Acknowledgments

My thanks to Andrew Bond for suggesting a system like
CRIB, to CRIB design team leader Frank George for in cluding me in the team, and to Les Rabbitts of ICL engi neering training for providing some of the early ideas. I also extend my thanks to team members Bob Beakley and
Ted Newman of the advanced computer technology pro gram, or ACTP, and to Gerry Piper and Tony James of ICL for all of their work. I am especially grateful to Tom Addis of ICL RADC (now of Brunel University) for his hard work, constant encouragement and numerous in sights.

This article is an expanded version of a paper entitled "How Expert Should an Expert System Be?" presented at the Seventh International Joint Conference on Artificial Intelligence, University of British Columbia, Vancouver, Canada, August 1981.

References

I . M. R. Genesereth, "Diagnosis Using Hierarchial Design Models," *Proc. AAAI*, Carnegie-Mellon University/Uni versity of Pittsburgh, Aug. 1982, pp. 278-283.

2. H. Shubin and J. W. Ulrich, "IDT"-An Intelligent Diagnostic Tool," *Proc. AAA]*, Carnegie-Mellon Univer sity/University of Pittsburgh, Aug. 1982, pp. 290-295.

3. J. McCarthy, "Epistemological Problems of Artificial In telligence," *Proc. Fifth Int'l Joint Conf. Artificial In telligence, MIT*, 1977, pp. 1038-1044.

D. W. Hamlyn, *The Theory of Knowledge*, Anchor Books, Garden City, New York, 1970,

5. E. A. Feigenbaum, "Themes and Case Studies of Knowledge Engineering," in *Expert Systems in the Microelectronic Age*, Donald Michie ed., Edinburgh University Press, Edinburgh, Scotland, 1979, pp. 4-25.

6. L K. Schubert, "Extending the Power of Semantic Networks," *Artificial Intelligence, Vol.* 7, No. 2, 1976, pp. 163-198.

7. L. Johnson, *Practical Reasoning*, Brunel Univ., ManComputer Studies Group, tech. report M-CSG/TR/1, **1979** (available from the author),

8. T. R. Addis and R. T. Hartley,A *Fault Finding Aid Using a Content Addressable File Store*, ICL Research and Advanced Development Centre, tech. report 79/3, Stevenage, U.K., 1979.

9. F. Hays-Roth, D. A. Waterman, and D. B. Lenat, "Principles of Pattern-Directed Inference Systems," in *PatternDirected Inference Systems*, D. A. Waterman and F. Hays-Roth, eds., Academic Press, 1978, pp. 576-601.

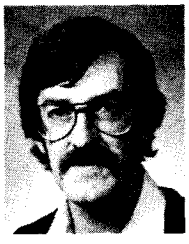
10. T. R. Addis, "Towards an 'Expert' Diagnostic System," *ICL Tech. J., Vol. 1*, No. 1, May 1980, pp. 79-105.

11. E. Babb, "Implementing a Relational Database by Means of Specialized Hardware," *ACM Trans. Database Systems, Vol.* 4, No. 1., 1979, pp. 1-29.

12. R. Duda, J. Gaschnig, and P. Hart, "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," in *Expert Systems in the Microelectronic Age*, Donald Michie ed., Edinburgh University Press, Edinburgh, Scotland, 1979, pp. 154-167.

13. W. J. Clancey, "Tutoring Rules for Guiding a Case Method Dialogue," Int'l J. Man-Machine Studies, Vol. 11, No. 1, 1979, pp. 25-49.

14. W. J. van Melle, System Aids in Constructing Consultation Programs, UMI Research Press, Ann Arbor, Mich., 1981.



Roger T. Hartley is an assistant professor in the Department of Computer Science at Kansas State University, where he is currently researching expert systems methodologies and is engaged in the design of generic planning systems and description languages. He has been a lecturer in both cybernetics and computer science in England and was a consultant to International Computers Limited. He is a member of ACM, AAAI, SAISB (the Society for the Study of Artificial Intelligence and Simulation of Behavior). Hartley received his PhD in cybernetics from Brunel University in England in 1974.

Hartley's address is Kansas State University, Dept. of Computer Science, 121 Fairchild Hall, Manhattan, Kansas 66502.