Acquiring Knowledge for Validation Models

Evangelos Simoudis James Miller[1]

Digital Equipment Corporation
Cambridge Research Lab

CRL 90/5 November 3, 1990

Abstract

We present a methodology for building validation models, the knowledge structure used during the retrieval phase of our case-based reasoning systems. This methodology allows a knowledge engineer to start with a textual database of previously solved cases and quickly produce an effective case-based reasoning system. Using our methodology, we have built two such systems; one of these systems operates on a domain spanning 60% of the problems presented to an existing rule-based expert system. By comparison, the rule-based system took four times as long to construct at an estimated cost six times as great. While our system successfully solves 100% of the problems in its domain, the rule-based system is only capable of solving 40% of the problems in the full domain.

[1]The authors wish to thank David Waltz for his help with this research.

1 Introduction

We are studying retrieval in the context of case-based problem-solving. Our goal is to utilize existing databases as case bases, retrieving all the cases that are relevant to a new problem, while retrieving as few as possible of the irrelevant ones. We employ a two-step retrieval process: retrieval based on surface features (SFB retrieval), followed by validation of the retrieved cases. The first step uses easily acquired information about surface features, but it is not sufficiently discriminating. In our studies, 11% and 7.6% of the cases in databases were returned in each SFB retrieval. The second step, validation, uses knowledge about tests that can be performed on the new problem. We have proposed[13] encoding this knowledge in a validation model. The combination of SFB retrieval and validation reduces the number of cases retrieved to 1.5% and 0.58% respectively, without omitting any relevant cases.

In this paper, we present a methodology for building validation models. This methodology allows a knowledge engineer to start with a textual database of previously solved cases and quickly produce an effective case-based reasoning system. Using our methodology, we have built two such systems. The first of these systems, CASCADE, operates on a domain spanning 60% of the problems handled by an existing rule-based expert system, CANASTA. By comparison, CANASTA took four times as long to construct as CASCADE at an estimated cost six times as great. While our CASCADE system successfully solves 100% of the cases in its domain, CANASTA is capable of only solving 40% of the cases in the full domain.

Our methodology reduces the cost of building the system by allowing the knowledge engineer to begin the knowledge acquisition task without contacting a domain expert; instead, the engineer reads the existing database. Based on this, and other background reading, the engineer prepares a proposed validation model. The engineer and the expert then meet (for the first time) to begin the iterative process of refining the validation model. The process terminates when the expert agrees that the model is correct and the engineer agrees that it has sufficient detail to encode the cases from the original database. Finally, the engineer encodes each case from the database using the validation model, producing a working expert system.

2 Related Work

A number of other case-based reasoning systems use retrieval based on surface features followed by a justification phase. These systems use the most common form of deeper knowledge, causal models, for reasoning during justification. While CASEY[8] uses a pre-existing causal model, CHEF[6], PRIAR[7], KRITIK[5], CYCLOPS[14], and PROTOS[2] all develop their own causal models for their domains. None of these systems addresses the knowledge acquisition problem involved in building a causal model. PROTOS, however, uses a causal model to facilitate the acquisition of classification knowledge. Classification knowledge is more simply structured than causal knowledge, and is thus easier to acquire. In fact, we suspect that the designers of PROTOS would agree with us: it is hard to acquire the knowledge needed to build a causal model. Unlike these systems, we entirely avoid the problem of building a causal model. Instead, we propose the acquisition of validation models; these models are much simpler to construct than causal models (but still considerably more difficult than the simple classification knowledge of PROTOS).

PRIAR and KRITIK use causal explanations during case reasoning to adapt existing solutions to fit new problems — an important area, but one that our work does not explore. CYCLOPS concentrates on the reverse of our problem: we are concerned with retrieving too many cases for careful scrutiny, while in CYCLOPS the concern is that no relevant cases may be retrieved. CYCLOPS uses deep knowledge to modify the indices used during the retrieval phase, an approach that meshes nicely with ours to produce an automated method for improving the performance of SFB retrieval.

In addition to the work of the case-based reasoning community, our work is related to that of the knowledge acquisition community. Tools for the automatic acquisition of knowledge have concentrated on specific types of knowledge (for example, Bennett's ROGET[3] concentrates on application domain conceptual structure and Minton's PRODIGY[12] concentrates on control knowledge). In addition, Davis's TEIRESIAS[4], Anderson's GEOMETRY TUTOR[1], Newell's SOAR[9], and Mitchell's LEAP[11] refine and compile (rather than acquire) existing knowledge. Our work, however, attempts to improve the overall task of the knowledge engineer. In fact, we do *not* provide an automation tool, but rather define a more efficient methodology which we hope will subsequently be aided by automated tools.

3 Acquiring Knowledge: Our Viewpoint

We are particularly concerned with the engineering aspects of knowledge-based systems: building an efficient system in a reasonable amount of time and at an acceptable cost. At present, the cost of creating production-quality systems is high and they take too long to construct. Despite the proliferation of rule-based expert systems, implementing each such system remains very much an art that is performed by a knowledge engineer in conjunction with a domain expert. As with other programs, rule-based systems are dominated by a large number of components designed to handle special situations, and are plagued with interdependencies between these and other components.

We consider two problems, *time* to build and *cost* of building, separately; the dominant factor is different in the two cases and the solutions may well be different. The time to build an expert system is dominated by the efficiency of the knowledge engineer, and can be improved by such things as expert system shells. The cost, however, is often dominated by the amount of time required of a domain expert. These experts are hard to locate, their time is often overcommitted to priority projects elsewhere, and they are very highly paid. A reasonable estimate is that a domain expert costs eight to ten times as much, per day, as a knowledge engineer when all of the ancillary costs are considered.

Fortunately, knowledge acquisition is at the heart of *both* of these problems. Our methodology provides a specific knowledge representation scheme (the validation model), thus removing one major design decision from the knowledge engineer's concern, and consequently increase productivity. We significantly reduce the costs by using an existing database of cases (prepared by domain specialists[1] and experts as part of their routine work) to allow the knowledge engineer to acquire domain knowledge without direct assistance from an expert. Furthermore, the knowledge engineer's use of the database enables knowledge acquisition sessions to be very carefully focussed, again reducing

[1]In both of our applications we have identified individuals who have specialized knowledge of the domain but cannot be classified as experts. In general, these people have operational knowledge of the area, but don't necessarily understand complex interactions or causality within the domain. For example, most general practitioners of medicine would qualify as domain specialists: they have sufficient general knowledge to diagnose common problems, but they lack the knowledge needed to handle rare or specialized problems. We find it useful to distinguish specialists from experts because of the difference in impact on the cost equation: specialists may be costly, but experts are downright expensive.

the expert's time and consequent costs.

4 A Tale of Two Systems

This section compares the time and cost of developing two different expert systems, both for diagnosing crashes of the VMS operating system. The first system, CANASTA, is rule-based and was developed by another group within Digital; it attempts to handle all forms of crashes. We developed the second system, CASCADE, concurrently but independently, using the methodology described in this paper. It is case-based, and is designed to handle only crashes related to device driver failures, which account for 60% of the problems encountered in practice. Section 6 presents additional details of CASCADE, as well as a second case-based system we have developed.

Figure 1 compares CANASTA to CASCADE, showing a breakdown of the

	CANASTA	CASCADE
Method	265 rules	200 cases
Domain	VMS crashes	VMS crashes
Subdomain	All	Device-Driver[1]
Success Rate	40%	60%
Acquisition time (pdays)	80[2]	20[3]
Encoding time (pdays)	600	85[4]
Maintenance time (pdays)[5]	280	0
Total time (pdays)	960	105
Total cost (est.)[6]	1600	205

1. Crashes related to device-drivers account for 60% of all crashes reported for VMS.
2. Spent with a domain expert.
3. 5 pdays spent with a domain expert and 10 pdays with a domain specialist.
4. This includes one-time development that would not be repeated for another expert system using the same methodology.
5. See the text for a definition of maintenance time.
6. This estimate is based on a domain expert costing eight times a knowledge engineer and a domain specialist as four times a knowledge engineer.

Figure 1: Two Expert Systems: CANASTA and CASCADE

overall development cost into three disjoint components: knowledge acquisition, knowledge encoding, and maintaining the knowledge. By "maintaining the knowledge" we mean, precisely, the time required to merge two self-consistent sets of rules (an initial working set of rules and a set of rules required to encode newly acquired knowledge) to produce a self-consistent whole. Notice that our definition of maintenance is *not* the traditional distinction between time spent prior to shipping an initial version versus that spent later; rather we are apportioning all of the time spent (before or after shipping) into one of three disjoint tasks. Figure 1 shows that the cost of developing CANASTA, the rule-based system, was 7.8 times higher than for CASCADE, and the development time was over 9 times as long.

These development times need to be examined carefully, however. There are two contradictory factors that must be considered. On one side, the savings may be even larger than these figures indicate. Since CASCADE was the first system we developed using our methodology, part of the 85 pdays spent encoding knowledge includes the initial design of the data structures used for the validation model. This design time is not repeated for later systems, since the same basic data structure are used for all validation models[2].

On the other hand, it isn't quite fair to reason that "Since CASCADE handles 100% of the problems in its domain which includes 60% of the problems in CANASTA's domain, it has a effectiveness of 60%. Since CANASTA has only a 40% effectiveness, CASCADE outperforms CANASTA." Nor should the development time or cost ratios be merely multiplied by 0.6 to estimate the cost of using our methodology to produce a "full" system. It may well be true that the knowledge needed to decide whether or not a problem falls into CASCADE's domain is, in fact, the hardest knowledge to acquire — thus, our methodology may not scale in this sense. At the same time, our ability to use an existing database allowed us to recognize that 60% of the cases are related and can be easily transformed into a single highly reliable expert system. It is possible that the remaining 40% of the cases can be easily encoded into one or more additional expert systems, and a new problem could be submitted to all the systems concurrently to locate relevant cases. We have not explored this area.

Notice that we have included maintenance time for CANASTA but we have no comparable number for CASCADE. This is not accidental. Recall our definition of maintenance: it is the time taken, above and beyond knowledge

[2]With our 20/20 hindsight we now realize that we should have measured this time separately.

acquisition and encoding, to add new knowledge to a system. In our case-based system there is no such additional time. In a rule-based system, the addition of rules take place in what amounts to an imperative programming language where the sequencing of rule firing is of primary importance. Largely because of the imperative nature of the rules, addition of a rule often requires modification to other rules in order to maintain an acceptable ordering to rule execution. It is this cost that we call "maintenance," and it can be quite expensive. In CANASTA, maintenance already accounts for 17.5% of the cost after only 6 months of use — and this maintenance cost rises with the number of rules, and hence with time.

We now return to the major concern of this paper, knowledge acquisition. In the development of the rule-based VMS diagnosis system, 720 engineer-days of costs went into knowledge acquisition (120 pdays from the knowledge engineer and 120 pdays from the domain expert). In developing our system for the same domain, the comparable number is 120 engineer-days (20 for the knowledge engineer, 15 for the domain specialist, and 5 for the expert). The reduction in cost (a factor of six) is tremendous, as is the reduction in time (a factor of 4). The remainder of this paper explains how this reduction is accomplished: as we have already said, it arises from the use of both an existing database of analyzed cases (to supply data and aid in knowledge acquisition) and a validation model (to encode knowledge).

5 The validation model

We describe elsewhere[13] our two-stage methodology for case-based reasoning. It involves an initial SFB retrieval followed by a justification phase based on deeper knowledge. It is acquiring the domain-specific knowledge for this latter phase that concerns us in this paper. The knowledge we require describes tests that can be performed on new problems for comparison with results of similar tests that were performed on cases in the existing case base. We encode this knowledge in a data structure we call a validation model.

We now describe the validation model in a bottom-up manner. At the very lowest level it consists of individual actions that can be performed on new problems to produce data values. These actions correspond to the simplest kinds of tests that can be performed; for example: reading a value in a known location in memory, measuring the amount of fuel in the gas tank, or taking a patient's temperature.

At the next level, these actions can be grouped together into probes to yield less detailed knowledge about the new problem, but knowledge which is more descriptive and hence likely to be more indicative of the kind of difficulty that has been encountered; each action can participate in more than one probe. In the car example we might have one probe to determine if the "fuel injection is working" and another to detect whether the "fuel is low". Both of these probes will require using the action "measure the amount of fuel in the gas tank."

But probes do more than simply group together actions. They summarize the information provided by the constituent actions into a canonical form that can be compared with the summaries stored in the case base. The precise form of the summary is dependent on the purpose of the probe, so two probes that are composed of the same actions may yield very different canonicalizations. For example, in medical diagnosis we might have a probe designed to answer the question "does the patient have a fever?" and another for "is the patient hypothermic?" Both of these depend on the action of taking the patient's temperature, but they canonicalize this knowledge (into a boolean value) in very different ways — the first will be true for temperatures above 99.5°F, the second for values below 97°F.

Probes also produce "interesting information" that was learned in the process of computing the canonical value, and this information is made available to other probes as an information block. For example, the exact temperature or the exact amount of fuel in the gas tank might be made available in addition to the abstracted information ("fever," or "fuel low"). These information blocks are an explicit part of the validation model, and act as a cache for intermediate information. When a new problem is examined, all of the information blocks in the model are cleared, and they are filled in as probes execute. Each information block specifies a probe which can be executed to fill in its contents.

To accomplish all of this, probes actually consist of two components. The first is the ordered set of actions to be performed. The second is what we call the interpretation knowledge. This takes as input the values computed by the actions as well as the information blocks from earlier probes. It produces two outputs: the canonical value of the probe, and an information block for use by subsequent probes. The fact that one probe requires the information block from another probes implies an ordering to the execution of these probes. When a probe runs, it checks all of its input information blocks to see if they are filled with data from the current problem. If not, it runs the appropriate probe to fill the block. For example, we may wish to execute a probe to determine whether a patient has an infection. This requires both determining that the patient has a fever and that the white blood count is high. The former is a probe, and the information block filled in by the fever probe will convey the answer. The latter is composed of a sequence of actions ("take a blood sample," followed by "count the white cells").

So far, we have encoded a precedence ordering on the probes based on the information blocks coupling the output of one probe with the input of others. "Probe-space," however, is more complicated than this. It has a roughly hierarchical structure that is related to the conceptual structure of the system the probes are designed to test. That is, some probes correspond to complex concepts (like "infection") and others to simpler concepts (like "fever"). This conceptual structure is unrelated to the precedence ordering, and is not directly useful in justifying new problems. It facilitates both the knowledge acquisition sessions, by providing a structure to the discussions between the expert and the knowledge engineer, and the subsequent maintenance of the system, by allowing the knowledge engineer to quickly locate relevant parts of the validation model. The validation model actually consists of this **conceptual structure**, each of whose leaves is a set of probes.

Finally, we turn for a moment away from the validation model itself and examine the case base. Each case consists of three components: the surface features used in the earlier retrieval phase, the preferred solution for this case, and what we call the **validation procedure** which can be used to ensure that the solution for this case is applicable to a new problem. This validation procedure consists of a set of probes to be executed and the canonical values of those probes (each pair of a probe with its canonical value is called a **validation step**). In a world of perfect knowledge, these probes would form a complete causal chain. In fact, they would provide assurance based on first principles that *if* the probe values of a new problem result in these canonical values, *then* this case is relevant to the problem under consideration. The world, however, is not perfect[3], and we can make no such claim about the actual set of probes. The set of probes (and their canonical values) is chosen by the knowledge engineer from comments found in the original database from which the case base is derived.

[3]This observation, for which we claim no credit, is one reason we believe that causal models are poor candidates for justification in case-based reasoning.

5.1 Creating a validation model

With this background, our knowledge acquisition problem is clear: the knowledge engineer starts with a database of textually described cases, each of which includes a list of surface features, a set of tests that were performed to determine a solution, and the solution itself. The job is to produce a case base with the same surface features, and a set of validation steps (probes and canonical values of those probes), and a solution. That is, the knowledge engineer must discover the set of probes (actions and interpretation knowledge) implicit in the database, and then encode each case using this information. Stated another way, the knowledge engineer must first develop a validation model and then encode the cases using that model. The following steps encapsulate a methodology which we have found to be very effective for managing the knowledge acquisition process when trying to develop a new validation model:

1. **Read** each case and build a list of the tests that are explicitly mentioned, along with values derived from these tests. In the process of reading the database and preparing this list, a sense of the underlying (but unstated) relationships between tests is developed. This sense will help determine the probes that are needed for the validation model, the contents of their information blocks, the ordering of the probes, and a reasonable set of canonical values.

2. **Examine** the list of tests, attempting to form related sets of probes. This organization eventually becomes the conceptual structure of the validation model.

3. **Refine** the conceptual structure. This is done initially by consulting standard reference works about the domain, subsequently by interactions with domain specialists, and finally by interactions with domain experts.

4. **Iterate** the above two steps. The final validation model consists primarily of entries corresponding directly to information that appears in the original database. As this iteration proceeds, some items will move from probes to conceptual structure or vice versa; some will move from probes to actions and vice versa; and the precedence ordering of probes will become clear.

5. **Integrate** the probes into the conceptual structure, forming the final validation model. This model should be sufficiently clear that the domain

expert can examine it and provide verification that it appears correct.

5.2 Extended Example

Let's take a very simple example of a case database and trace through our knowledge acquisition methodology. We'll work in the domain of automotive diagnosis[4]. Assume a database consisting of the following three cases:

		CASE 1			
make:	MAZDA	**model**:	626	**model year**:	1988
engine type:	2.0L EFI	**miles**:	10,000		
problem:	engine does not start.				
validation:	The fuel injector was clogged.				
solution:	cleaned the fuel injector.				
		CASE 2			
make:	MAZDA	**model**:	626	**model year**:	1984
engine type:	2.0L	**miles**:	60,000		
problem:	engine does not start.				
validation:	The car had a faulty gas pump.				
solution:	Replaced the gas pump.				
		CASE 3			
make:	MAZDA	**model**:	626	**model year**:	1987
engine type:	1.8L	**miles**:	20,000		
problem:	engine does not start.				
validation:	A leak existed in the gas line.				
solution:	Fixed the leak.				

Before we begin knowledge acquisition, we partition the information in each case into surface features and deeper knowledge. In this example, the fields `make`, `model`, `model year`, `engine type`, `miles`, and `problem` constitute (we assume) surface features. Only `validation` is deeper knowledge that we must encode as a validation model. We begin by **read**ing each case, and develop the following list (to the left of the arrow ($\rightsquigarrow$) we list the text from the case, to the right we list our assumption about an appropriate test to perform and canonical value for that test):

[4]For a more elaborate example from the same domain, listen to "Car Talk" on National Public Radio.

1. The fuel injector was clogged $\rightsquigarrow$ `condition-of-fuel-injector`; `clogged`, or `normal`
2. The car had a faulty gas pump $\rightsquigarrow$ `condition-of-gas-pump`; `faulty`, `normal`
3. A leak existed in the gas line $\rightsquigarrow$ `condition-of-gas-line`; `leak`, `normal`

Here, we've simply assumed that we'll figure out how to probe for each of the conditions mentioned in the case base, and taken a guess (based on our command of English) about the results of such a probe. We don't yet know how to actually perform the probes, nor are we sure that these are the real values that one would discover.

The next step in our methodology is to **examine** the list and attempt to build a conceptual structure. In this example, unfortunately, there really isn't any obvious structure. We do notice, however, that a car has at least three components: a fuel injector, a gas pump, and a gas line. Furthermore, we know (or at least conjecture) that we will have one probe for each of these components. We still have no clue as to the actions needed to build our probes. We build a conceptual structure that has these three components, and has one probe for each of them.

Step 3 is to **refine** this structure. We go to the local book store and purchase a copy of Chilton's Guide for the make of our car. Our major interests at this point are figuring out how to implement each probe from simpler actions, and verifying the correctness of our conceptual structure . Our reading informs us that to determine the condition of the fuel injector the following actions need to be performed (and in this order!):

1. Open the hood of the car.
2. Unscrew the cover of the fuel injector.
3. Remove the nozzle that connects the fuel injector's reservoir with the combustion chamber.
4. Pour water through the nozzle and observe its flow.

The first three of these steps are straightforward, but the fourth one is rather hard to interpret. We need to know just how much flow constitutes (our postulated) `clogged` or `normal` values. For this, we **iterate** by locating a (hopefully friendly) domain expert. After a short discussion, we discover that the expert really has three, not two, values, and add `restricted` as an interesting possibility. We also find out what are reasonable rates of flow that correspond to each of these values.

In the course of this same conversation (or maybe during a later iteration), we discover that there is additional information gained by unscrewing the cover of the fuel injector: we can see if there is any fuel in the reservoir that feeds into the nozzle.

In a similar manner, we proceed to build up a collection of actions that constitute our set of three probes (as it happens, we picked the correct set of probes initially — this doesn't happen so often in practice). We also learn about the precedence constraints between the probes and their interconnection by information blocks.

After a sufficient number of iterations, we will converge on a conceptual structure that is sufficiently detailed for our needs and which a domain expert can verify as matching his understanding of the overall system. We then **integrate** this structure and our collection of probes, to form the full validation model. It is probably wise to have the domain expert verify this final model as well as the conceptual structure on which it is based.

We can continue just a small bit further by considering a knowledge maintenance problem. At a later date, we might become interested in the combustion chamber mentioned in our third action. All we need to do is to modify our conceptual structure to include the chamber — a simple, localized change. In doing this, we will probably realize that the amount of fuel in the injector's reservoir is important for probing the combustion chamber. This is easily encoded: we create an information block that allows the existing probe to propagate the amount of fuel to new probes associated with the combustion chamber.

5.3 How it helps

We have claimed that our methodology reduces the time to produce an expert system by simplifying the knowledge acquisition process. We have shown figures to support this claim, and we have explained the methodology itself. This section discusses a slightly more abstract issue: which aspects of our methodology simplify the problem?

First, our methodology doesn't attempt to automate the knowledge acquisition process (Section 7 discusses some ideas for automating part of the process). Instead, it aims at making the process more efficient, especially with regard to the amount of time required of a domain expert. We do this by significantly altering the role of the expert in the knowledge acquisition process. Essentially, we gain leverage off of the existing database. By having a knowledge engineer peruse this database and other knowledge sources, we can have the engineer produce a proposal for a conceptual structure of the expert's domain, based on the expert's own vocabulary, as recorded in the cases. Just as it is far easier to critique and edit a document than to create one, it is similarly easier for the expert to examine the proposed structure than to create one from scratch. Thus, we simplify the task that we ask the expert to perform exclusively for making the expert system. (We assume that the expert's normal duties include maintaining the database, and we exclude this time from our consideration.) It would be reasonable to expect *any* case-based methodology to simplify the knowledge acquisition problem in this way (in fact, this is an early claim in support of these methods). In practice, however, other case-based reasoning systems continue to rely on causal models whose acquisition remains very costly.

Secondly, we focus the attention of the knowledge engineer by providing specific knowledge structures (probes, actions, information blocks, conceptual structure). The case database also provides the initial seed for all of these structures. This point is important: in most design methodologies, the initial steps are the hardest to take, require the most thought, and often have the greatest penalty attached to small errors. Thus, any reasonable person (including a knowledge engineer) is very hesitant about taking them. Our methodology starts very simply: read the existing database, and write down the tests you see being used. The method of iterative improvement gradually guides the engineer's steps toward a working system structure, rather than requiring a carefully designed initial structure. In addition, our methodology has a self-contained measure that indicates when the process should stop: when the validation model contains all the probes needed to encode the existing cases, and a domain expert agrees with the overall precedence constraints.

Finally, we will raise an issue not directly related to knowledge acquisition, but perhaps the most vital in calculating the cost of any expert system: the problem of maintenance of the knowledge. The validation model is inherently structured, since it provides probes for grouping actions, a conceptual hierarchy to organize probes, and explicitly represents dataflow constraints through the use of information blocks. But modularizing constructs alone are only of theoretical interest; our methodology guides the knowledge engineer toward a practical use of the constructs. Since the validation model is repeatedly examined by domain specialists and experts, it gradually becomes structured in a manner they find acceptable for explaining their domain. On rare occasions, a major breakthrough may alter the experts' own perception of the domain's modularity, but these breakthroughs are extremely rare and will invalidate any representation of domain knowledge (consider the invention of the telegraph with respect to international banking). Much more often, however, the knowledge maintenance problem is really one of making a small addition or modification to the knowledge base. If the base is organized in a way compatible with the experts' view of the domain, then the change will almost certainly be relatively localized. It may require an expert to state where the change should occur or the precise nature of the change, but the fact that the place exists (and hence can be located) is an immense aid in the maintenance process.

6 Results

We have implemented two different expert systems using our methodology. CASCADE, for diagnosis of VMS crashes related to device drivers, was mentioned earlier and contrasted to the rule-based system CANASTA for a related domain. The second is for diagnosis of problems with the WPS–PLUS word processor. This section contains details of the implementation times, knowledge acquisition times, and performance of each system. In order to fully present each system, we provide information about both the initial retrieval based on surface features and the subsequent justification using a validation model.

6.1 Device Drivers

Information about surface features was obtained primarily from DEC internal publications and was complemented by a domain specialist from the VMS support team during three knowledge acquisition sessions. It took a total of 11 pdays to acquire the domain specific knowledge about surface features. This knowledge was implemented in five days.

It took an additional four days of reading the database and interacting with domain specialists to develop a validation model. In addition, four knowledge acquisition sessions with a domain expert, lasting five days, were needed to refine and improve the validation model. Encoding the actual validation model took 80 days. The total number of days spent on knowledge acquisition (KA) and development is shown below:

Activity	Person Days
KA (expert)	5
KA (specialist)	15
Development	85

The system was evaluated using a case base of 200 cases that were obtained from notes written by specialists. The SFB retrieval phase of the system was evaluated by presenting each of the 200 cases to the retriever (as new problems). UNIMEM[10], the algorithm that we have used for SFB retrieval provides a mechanism, known as **retrieval weights**, for tuning its retrieval capabilities. After some experimentation, we discovered that the use of larger retrieval weights (i.e. more stringent matching criteria) caused the retriever to miss many relevant cases and, on many occasions, to fail to retrieve any cases at all. With less stringent criteria this problem was rectified. However, many of the retrieved cases were not relevant to the problem. With the best weightings we found, we were able to retrieve on average 22 cases per retrieval (11%). The validation phase, however, was able to reduce this number of cases to an average of 4.5 cases out of 200 (2.25%).

In addition, we presented three new cases to the system. Based on SFB retrieval alone, we found 20, 25, and 16 cases (10, 12.5, and 8% selectivity). The validation phase reduced this to 3, 5 and 3 cases, respectively (1.5, 2.5, and 1.5% selectivity). Our experts confirm that these validated cases are the only ones relevant to the problems presented.

6.2 A Word Processing System

The second system performs diagnosis of customer problems with the word processing component of an office automation product. Information about surface features was acquired from a domain specialist and from internal publications within five days. It then took an additional five days to encode this information and use UNIMEM to develop a generalization hierarchy for SFB retrieval. We use a database with 340 cases.

The validation model was obtained from the cases in the database, an internal publication, and five days of knowledge acquisition with a domain expert. It took 40 days to implement the validation model.

The time we spent on knowledge acquisition and development is shown below:

Activity	Person Days
KA (expert)	5
KA (specialist)	5
Development	40

We have evaluated the system using the same method as in the first prototype. Our experiments showed that on an average 26 cases are retrieved during SFB retrieval, a 7.6% selectivity. The validation phase reduced this to an average of two cases, or 0.58% selectivity.

7 Automating the Acquisition Process

Our current methodology, while effective, is entirely manual. Because of the complexity of validation models, and because the input to our process is expressed in highly technical language, we do not expect to be able to automate the entire process. At this time, we have not yet explored any automation methods, although we see four very promising avenues to be pursued.

Corpus-based NLP.

The first step of our methodology is to read the entire database. Our largest database contains 340 cases described in about 500 pages of clean text; the initial reading of this database required five days. As the database becomes larger, this task becomes harder. By extracting the textual descriptions of how a case is validated and comparing this text with a standard English corpus (perhaps augmented by standard computer science jargon), we should be able to automatically produce a list of words that are good candidates for consideration as names of concepts, probes, or actions. If the NLP tool can refer each word back to a syntactic category and/or to the source text in which it occurs (along with frequency counts), this would be even more helpful. It would be unreasonable to replace the scan of the database entirely by automated tools, however, since part of the purpose of that scan is to help the knowledge engineer begin to understand the domain before the initial meeting with a domain expert.

Deducing precedence constraints.
If we make the (reasonable but not necessarily correct) assumption that the text of a case indicates tests to be performed in the order in which those tests are actually performed, we can combine these ordered sets to produce a proposed partial ordering of all the probes mentioned in the database. For each proposed ordering constraint the knowledge engineer would then produce an information block and ask whether data actually flows between the probes. (It is inherent in our model that the precedence constraints between probes are derived from dataflow constraints. There may be other types of constraints needed in other domains: costs of tests or destructive tests, for example.)

Deducing conceptual structure from probe sets.
In a process reminiscent of UNIMEM's concept formation, we can take the set of probes associated with each case and look for commonly occurring subsets. Each such subset can then become a candidate for forming a higher level concept, and the new concept replaces the earlier subset in appropriate cases.

Deducing conceptual structure from action sets.
A slightly more complicated process that extends the previous idea attempts to form new probes from commonly occurring subsets of *actions* within existing probes. This is more complicated because a probe uses its inferential knowledge to combine the results of the actions to form a single canonical result. In order to successfully combine common actions into a single probe, we would have to be sure not only that the actions all occur together in existing probes, but that the inferential knowledge combines the results in the same way in those probes. Thus, the same subset of actions might be combined into multiple *distinct* probes, based on how the action results combine to form a canonical result.

8 Conclusion

We have presented a methodology that dramatically reduces both the time and the cost of producing an expert system. In addition, the maintenance time associated with a rule-based system (the time required above and beyond that needed for acquiring and encoding new knowledge) is reduced to a negligible level. In one comparison, we reduced the time required to produce an expert system by a factor of four and the cost bya factor of six.

These improvements arise from three aspects of our methodology:

1. The methodology changes the roles of the knowledge engineer and the expert. The knowledge engineer begins by reading an existing database of solved problems and summarizing the information in the form of a conceptual model. The expert is asked to evaluate this conceptual model and provide feedback to the knowledge engineer. Thus, the engineer plays a much more active role in the process than in traditional rule-based system development, and the expert's time is more effectively utilized.

2. The methodology specifies the type of knowledge which must be assembled: information about tests that can be performed (actions and probes), and the relationships between them (conceptual model and dataflow constraints). This not only focusses the energy of the knowledge engineer, but it provides us with hope that part of the acquisition process can be automated. Since most of the work is centered around examining an existing database, we can use automated tools to examine the database for specialized terms that are good candidates for inclusion in the conceptual framework of the validation model.

3. The validation model consists of a standard set of data structures, so the knowledge engineer does not need to design data structures. This eliminates one major barrier in knowledge engineering: that of representing the acquired knowledge. Another major barrier, more psychological than technical, is also eased. Since our methodology is one of iterative refinement, it isn't necessary to aim immediately for "the correct structure" of the concept space. Any reasonable initial structure will serve well; this reduces the activation energy needed to get the implementation process under way.

We see three major directions for our future work. The first is to begin automating some of the steps in the methodology, along the lines discussed in Section 7. The second is to look at additional domains, farther afield from computer science, and verify that our methodology works well when the knowledge engineer is less familiar with even the high-level structure of the application domain. We are particularly interested in domains, like law or psychoanalysis, where there is reason to believe that a causal model cannot be constructed; we conjecture that case-based reasoning can work even in these domains. We are also interested in exploring our methodology when applied to problems other than diagnosis: we have built scenarios involving sales and management that appear to be amenable to case-based reasoning. Finally, we are interested in examining significantly larger databases — perhaps a database with 2,000 to 3,000 entries. As the number of cases grows, our methodology must certainly receive automated help (no engineer is likely to have time to read all of the cases). But, more importantly, it will almost certainly be necessary to deal with experts from different areas. Will it be most efficient to subdivide the cases to correspond to the different areas (as we've done by considering only device-driver failures in VMS), or will such a subdivision become unwieldly? How hard will it be to develop a single conceptual model that experts from different disciplines can understand (for example, imagine the validation model for an airplane where experts range from mechanical, to material, to electrical, to aerodynamic).

References

[1] John R. Anderson. *The Architecture of Cognition.* Harvard University Press, 1983.

[2] Ray Bareiss, Karl Branting, and Bruce Porter. The role of explanation in exemplar-based classification and learning. In *Proceedings of Case-Based Reasoning Workshop*, 1988.

[3] J.S. Bennett. Roget: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Automated Reasoning*, 1:49–74, 1985.

[4] Randall Davis. Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 12:121–157, 1979.

[5] Ashok Goel and B. Chandrasekaran. Use of device models in adaptation of design cases. In *DARPA Workshop on Case-Based Reasoning*, pages 100–109, 1989.

[6] Kristian Hammond. *Case-Based Planning: An Integrated Theory of Planning, Learning, and Memory.* PhD thesis, Yale University, 1986.

[7] Subbarao Kambhampati. Representational requirements for plan reuse. In *DARPA Workshop on Case-Based Reasoning*, pages 20–23, 1989.

[8] Phyllis Koton. *Using experience in learning and problem solving.* PhD thesis, Massachussetts Institute of Technology, 1988.

[9] J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[10] Michael Lebowitz. Experiments with incremental concept formation: Unimem. *Machine Learning*, 2:103–138, 1987.

[11] T. Michell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1, 1986.

[12] Steven Minton. Qualitative results concerning the utility of explanation–based learning. In *Proceedings AAAI-88*, pages 564–569, 1988.

[13] Evangelos Simoudis and James Miller. Validated retrieval in case–based reasoning. In *Proceedings AAAI-90*, pages 310–313, August 1990.

[14] Katia P. Sycara and D. Navinchandra. Index transformation and generation for case retrieval. In *DARPA Workshop on Case-Based Reasoning*, pages 324–328, 1989.