INTEGRATING TECHNIQUES FOR CONCEPTUAL MODELING

Timothy C. Lethbridge
Doug Skuce

Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5
(613) 564-8155 tcl@csi.uottawa.ca
(613) 564-5418 doug@csi.uottawa.ca

ABSTRACT

We describe seven knowledge engineering projects with which we have been involved. In these projects the knowledge represented is largely descriptive, has a significant informal content, and typically is creative as opposed to the being result of expertise-capture. We then compare eight major semi-automated techniques we have found most useful in the projects The techniques are: brainstorming, interpretive structural modeling, text scanning, conceptual diagraming, conceptual outlining, table completion, systematic critiquing and extended inference. Many of these techniques are widely known, but we have tailored them for our purposes. We suggest criteria for analysing the techniques and describe the techniques in terms of the these criteria. We also indicate how our knowledge management system, CODE4, has been developed with these techniques in mind in order to effectively support our knowledge engineering projects.

1. INTRODUCTION

Our goal is to facilitate the conceptual modeling of a domain using a flexible knowledge representation schema and supporting software. To facilitate this we have built several knowledge management systems, the latest one being CODE4[1] (Skuce and Lethbridge 1992)(Skuce 1992c).

In the rest of this introduction we describe the type of knowledge engineering we do. In section 2 we describe eight major techniques and list some properties and advantages of each technique. We analyse several of our knowledge engineering projects to see what techniques were used and with what results. We describe how CODE4 has evolved to support these techniques in a highly integrated way, and where we see room for further automated assistance. Finally, we present some some heuristics that indicate what techniques seem most appropriate to what kind of project.

1.1 Our flavor of knowledge engineering

As emphasized by Bradshaw et al (Bradshaw, Chapman et al. 1992) it is becoming generally accepted that many aspects of knowledge engineering primarily involve

1 CODE stands for Conceptually Oriented Design/Description Environment. The current version is CODE4 which in a complete redesign of CODE2. The knowledge engineering projects described in this paper have used both versions. CODE3 was an experiment that was never used for practical knowledge engineering.

constructive modeling. The old emphasis on *extracting expertise* is often a secondary focus, and may not be present at all.

In our work, we also find that we primarily need to represent the following types of knowledge (which are typically highly correlated):

a) Declarative (i.e. non-procedural) knowledge.

b) Descriptive properties of things (i.e. not rules).

CODE4 and its predecessors are primarily designed, although not restricted, to handle these types of knowledge.

CODE4 reflects our increased understanding of the importance of:

• The knowledge representation: its comprehensiveness and expressiveness.

• User interface design for knowledge management systems.

• General knowledge management[2] techniques.

In this paper we focus on the techniques; this necessitates describing a few aspects of the user interface. The following paragraphs provide a few points about the knowledge representation so that the reader can better understand the rest of the paper:

The units of knowledge in CODE4 are termed concepts. These are categorized into: type concepts, instance concepts, terms, predicates, statements, rules etc. Concepts are the machine's model of anything it can reason about. Analogously we can think about concepts in the mind as representing anything the mind can think about.

CODE4 has a number of advanced knowledge representation features (Lethbridge 1991a). Of particular note is its ability to represent concepts on a formal-informal spectrum (Lethbridge 1991b), (Lethbridge and Skuce 1992). Informal knowledge is usually in the form of text strings that cannot be immediately interpreted by the system, and relies on interpretation by the user of the knowledge (people or other systems). Formal knowledge is tightly connected to other knowledge with links that have a well-defined semantics.

1.2 Recent knowledge engineering projects

The following paragraphs describe some of the major projects in which we have been involved, using CODE2 and CODE4. The list is not exhaustive: we have built numerous other small knowledge bases, and various other individuals and organizations use CODE for their own purposes, e.g. (Bradshaw, Holm et al. 1992)

- **The Bell-Northern Research Telos project**

  This project (Skuce 1992b) lasted 18 months and consisted of building a knowledge base about Telos, a complex software system under development. The effort included reverse engineering Telos as well as assisting in the conceptual design of new features. Two knowledge engineers and several software engineers and managers were involved.

---

2 By *knowledge management*, we include knowledge acquisition, the refinement of the knowledge, and making the knowledge available for use (i.e. as a server). In this paper we focus on the first two of these aspects.

- **The Telebrain project**

  In this project we designed a hypothetical 'invention', a new kind of telephone system. The focus was on allowing the knowledge engineer to be as creative as possible (as he or she might be if using traditional tools such as paper and pencil) while gaining the power of a knowledge representation system.

- **The Ontology project**

  [3]This on-going project (Skuce and Monarch 1990) is attempting to identify a useful set of general concepts for the top of an 'isa' hierarchy, based on linguistic research. The objective is to achieve a conceptually elegant ontology that as many people as possible can agree on, and use as a basis for exchanging knowledge bases.

  So far, we have developed a number of iterations of the ontology. Each time we have refined our thinking and incorporated the ideas of others (Lenat and Guha 1990), (Miller 1990).

- **The CODE self-description project**

  Several times during the development of CODE2, CODE3 and CODE4 we created knowledge bases describing our conceptual understanding of the system under design.

- **The Cogniterm project**

  This on-going project (Meyer, Skuce et al. 1992) is exploring how a knowledge management tool can assist the terminologist. Much of the focus is on defining terms and describing how they are used.

  The main knowledge base contains descriptions of several hundred concepts in the domain of 'optical storage media'.

- **The Smalltalk description project**

  As a new tactic in our strategy of using CODE for software development, Iisaka (Iisaka 1992) has created a knowledge base about the Smalltalk language (in which CODE2 and CODE4 are implemented). CODE4 has also been extended so that the knowledge base can make direct reference to Smalltalk objects.

2. HIGH LEVEL CONCEPTUAL MODELLING TECHNIQUES

Figure 1 is a 'knowledge flow diagram' showing the main techniques that we use. Knowledge entering by any of the techniques comes from the minds of people, from documents or from knowledge already existing in the knowledge base.

The subsection 2.1 describes some of the criteria by which we can categorize the techniques and subsection 2.2 describes the techniques in more detail. In section 2.4 we indicate how the various projects described above made use of the techniques.

---

3 The word 'ontology' is developing a variety of meanings. Here we use it in the sense of highly general concepts applicable to a wide variety of domains.

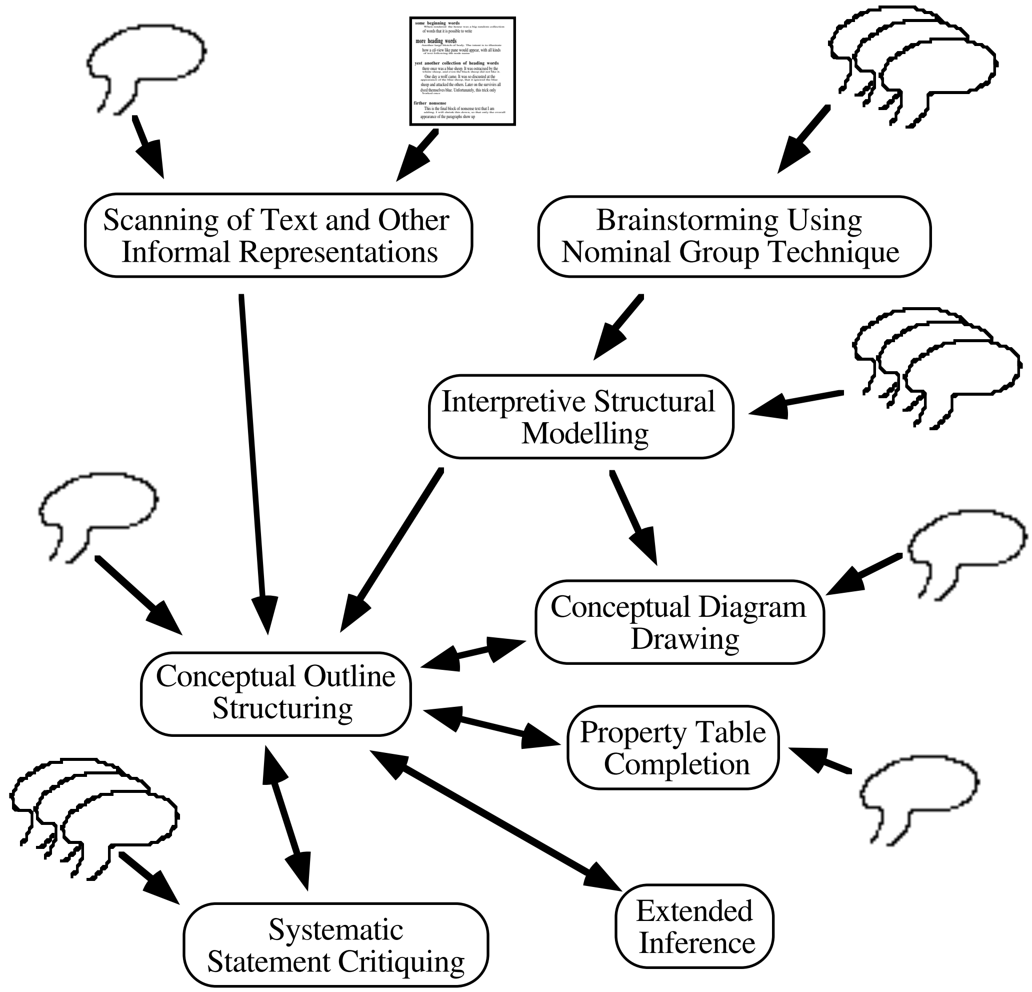

Figure 1: A knowledge flow diagram showing the various high-level knowledge acquisition techniques we use as knowledge is gradually formalized. The arrows represent the flow of knowledge between techniques (they do not indicate time sequencing since, for example, brainstorming may be used several times to expand the knowledge). Note that conceptual outline structuring plays a central role: we have found this to be the most generally useful technique.

2.1 General questions to ask about the techniques

The techniques in figure 1 all help build up a knowledge base, but they have very different properties. In particular we can ask the following types of question of each technique:

- **Pre-existing knowledge**

  What knowledge needs to be present prior to the technique being used? What is the starting point for the technique?

  Most techniques require existing knowledge that they can refine or use in structuring other knowledge. For some techniques this input knowledge needs to be relatively formal, whereas for others it is preferable that it be in the form of informal natural language statements.

- **Type of knowledge that results**

  Is the resulting knowledge new, or is it a refinement of pre-existing knowledge?

  It has been said (Skuce 1992a) that there are three main activities that can be performed by users of a knowledge management system: 1) acquiring new knowledge (often involving the consensus of several people), 2) critiquing and refining existing knowledge, and 3) querying, extracting and referring to the knowledge (using the knowledge base as a server). The first two of these are the knowledge acquisition activities that concern us here, although 3 is a constant need during 1 and 2. Some of the techniques are primarily oriented towards new knowledge, whereas others are oriented towards refining knowledge.

  Is the resulting knowledge informal or is it tightly connected to existing knowledge in the knowledge base (i.e. is it formal)?

  Some of the techniques result in informal (typically natural-language) statements or loosely structured knowledge, while others increase the linkages between concepts, making the knowledge base more formal.

  Answers to the two questions above are correlated: Techniques that generate new knowledge tend to generate informal knowledge. Techniques that refine knowledge tend to make it more formal.

- **Creativity and synthesis**

  Is the user free to enter whatever comes into his or her head, or is he or she constrained to answer questions or follow a machine prescribed procedure? Who initiates the next action, the user or the machine?

  Two attributes of a machine-assisted technique can answer this question: modality and interruptability. The former applies to fine grained actions in the user interface, the latter applies to switching back and forth between techniques.

  If the user interface is *modal*, the user must respond to prompts. Most user interfaces have more modality than is strictly necessary: For some techniques it is possible and desirable to remove all modality (the user is in complete control, no keystroke or movement of the mouse is requested by the machine); other techniques naturally require a modal interface.

  If the technique is *interruptable*, then the user can switch to another technique, suspending what he or she is doing in the current technique. This is generally good, but can be to distractive for some techniques.

  We hypothesize that less modality and greater interruptability have the potential to stimulate *creativity*: The user is not constrained to a particular course of action, and this is better able to represent whatever thought come into her or his mind. Greater interruptability also helps boost *synthesis*, since the user can go looking for knowledge to combine.

  What feedback should the user get to stimulate further ideas?

  Some techniques are more oriented towards giving the user freedom to enter knowledge, whereas others focus on displaying the consequences of what is entered. Feedback can be of three forms:

  1) *Group participant feedback*: Other participants in the technique see what has been entered and respond

2) Computer responsive feedback: The computer displays the consequences of user editing actions (in some systems this is done modally, with the user being forced to respond to the feedback, however we find it best not to force responses even if the user has entered something ‘wrong’)

3) Computer navigational feedback: The computer displays knowledge structures as the user navigates, often these knowledge structures must be dynamically computed. Typically the user will navigate around the knowledge base after every edit. The consequences of previous edits then become clear. This type of feedback differs from the last in that the user must specifically request it (although the request may be implicit).

- **Group efficacy**

  Is this technique suitable for use by a group?

  Some techniques are naturally group techniques. For these we would want to build ‘groupware’ user interfaces where several people can interact in parallel on their own screens. Other techniques are naturally oriented to individual knowledge acquisition. Some techniques are better performed on one screen (no parallel input) but can be effectively performed with a group of people guiding an operator.

In the next subsection we answer the questions in the context of each technique. The answers are summarized in table 1. In table 2, we also answer some of the questions in the context of the major projects in which we used the techniques.

The above questions are certainly not the only ones we can pose of the techniques. For example it might be useful to ask how good each technique is at producing problem-free knowledge (perhaps using the eight criteria listed under the systematic statement critiquing technique in the next section). This is a topic for future research.

2.2 Techniques amenable to computer assisted conceptual modeling

In this section we describe in detail the eight techniques shown in figure 1. We answer the questions posed in the last subsection and describe some of our experiences with the techniques.

The common threads that ties all the techniques together are: 1) they have been found effective for the type of conceptual modelling described in section 1.1, and 2) they can usefully be packaged together in an integrated computer assisted knowledge acquisition system, as indeed we are currently doing in CODE4.

- ***Brainstorming using nominal group technique***

  Starting point: A *trigger question*, of which participants are to think of as many answers as possible. No other knowledge need exist. This is decided by the group or moderator; it gives an essential focus to the session.

  Result: New, largely informal knowledge: a list of concepts that are answers to the trigger question. The concepts are typically expressed merely as text strings, although graphical output (sketches) might also be possible. The resultant concepts are typically a mixture of type concepts, instance concepts and statements; however it is up to a later technique to actually categorize them as such and to take other steps to formalize them.

  Technique: This technique is typically manual (Delbecq and al 1975), but recently computerized brainstorming assistance tools have appeared. Participants are arranged in a cycle and attempt to think of answers to the trigger question. As soon as they think of a possible answer it is made available to the next participant in the cycle. Similarly, answers from previous participants in the cycle stimulate new lines of thought, and hence answers.

Our insight is to treat the items that result from brainstorming as concepts in their own right. Such concepts would be very informally specified, but a system like CODE4 is capable of dealing with any level of formality. Later techniques would incrementally increase the formality of the knowledge. We envisage participants working at their own screens, with windows for input and for browsing of the ideas of others.

Interface modality: Fully non-modal. At any time, the participant is free to inject new answers, to browse previous answers etc.

Interruptability: Best not interrupted to use another technique because that would result in a participant effectively dropping out of the nominal group.

Feedback potential: High, but originating from the parallel efforts of others answering the trigger question. There is no opportunity for the system to comment on participants' input since it is too informal.

Group-efficacy: Requires a group.

Our experiences: In the BNR project we used this technique several times with no computer support, subsequently performing interpretive structural modelling (below). and then entering the results into a CODE2 knowledge base using conceptual outline structuring (see below).

We have also tried other less structured approaches to brainstorming, particularly conceptual outline structuring (below) with a large screen shared by a group. These approaches, while useful, do not generate as large an amount of new ideas and group participation as nominal group technique.

Example: When designing a subsequent version of CODE we held a brainstorming session with the trigger question, 'What features should we add in order to provide better support for requirements analysis?'. Some thirty ideas were generated by brainstorming. Many of these were not new, but some gave us fresh insight. The speed with which we could completely generate a more or less complete spectrum of ideas was particularly useful.

- ***Interpretive structural modelling (ISM)***

  Starting point: A list of more-or-less related concepts. These may be the result of brainstorming or informal representation scanning.

  Result: The starting concepts grouped, categorized or arranged in other hierarchies or arbitrary graphs according to some set of relations. Thus existing knowledge is refined and made more formal.

  Technique: Historically, a manual technique (Warfield 1976), but computerized assistance exists as exemplified by BNR's PRISM tool (Saunders 1990). The procedure varies depending on the number of concepts and the type of structure to be built, but the main ingredients are a well-structured and rapid sequence of moderated discussions and votes.

  Once again, our insight is to structure concepts directly using a knowledge based system. For example ISM can be used to classify concepts when building an isa hierarchy, or to structure conceptual relations such as temporal or importance graphs.

Interface modality: Largely modal. The group members respond to prompts that request them to vote.

Interruptability: Interruptable to perform brainstorming or some other technique, particularly when the group feels that the input concepts are inadequate to compose the structures being built.

Feedback potential: Typically low. It is possible to observe the structure being built, but this can distort the elicited knowledge. It is generally best to let the participants respond to the prompts by thinking about what they are being asked rather than attempting to look it up.

Group-efficacy: Most useful in a group, but can be performed individually.

Our experiences: We frequently perform this technique following the use of the nominal group technique to place some structure on the answers to the trigger question.

Example: After brainstorming for a list of concepts (representing features) we decided to categorize them. The first step was to take an arbitrary pair of concepts and ask the participants, 'Do these fall in the same category?' If there was disagreement a discussion was held. This was followed by a vote. This pairwise-voting procedure was repeated until the computer determined that it had sufficient information to categorize all concepts. After naming the categories we performed further ISM to arrange the concepts (including the new categories) in an 'is more important than' relation.

In this particular example, the categorization resulted in an informal 2-level isa hierarchy. Part of the informality was due to the fact that the concepts were still expressed as character strings. Additional informality resulted from the fact that the categories were often not really types or superconcepts of the input concepts. A purist knowledge engineer would in fact have been rather displeased at this initial attempt at creating an isa hierarchy. Nevertheless, *real knowledge was generated* and a knowledge engineer was subsequently able to rapidly clean up the knowledge. The categories provided a very useful breakdown of types of features, a breakdown that nobody would have individually thought of. Given the categories, it became possible to generate further knowledge by asking the question, 'What is missing from this partition?' during systematic statement critiquing (below).

- ***Scanning of text and other informal representations***

Starting point: A possibly-complex informal representation that cannot be fully parsed and compiled into a formal knowledge structure. Typically this is a natural language document or the description of a single concept using natural language. In some future implementation it might be possible to scan visual representations looking for patterns, icons etc.

Result: A series of new terms extracted from the input.

Technique: Much knowledge that we would like to capture in a knowledge base is often expressed in natural language documents. We believe that with current technology it is unreasonable for a knowledge-based system to attempt to 'understand' an entire text document.

There have been many attempts to extract terms out of documents *without* full natural-language understanding (Szpakowicz 1990). CODE2 can take a document and look for terms that are not contained in its lexicon. These are presented to the user as possible candidates for inclusion in the knowledge base (see figure 2). The user may accept the term (possibly declaring information such as part-of-speech), or reject it.

Figure 2: Text scanning of a natural-language document. The term 'PCM' has not been found in the Cogniterm knowledge base.

As an extension to the technique, the scanning process can pause on terms it already knows, presenting its current knowledge about the concept or concepts(senses) the term means. The user can then be prompted to confirm or declare the correct sense. As an future extension, built into a hypertext system, the system might tag the term in the input document with a pointer to the knowledge base.

Interface modality: Largely modal. The user is presented with a sequence of prompts requesting acceptance, rejection, confirmation, etc.

Interruptability: Interruptable at any time. In fact it is often desirable to suspend scanning and structure a concept when a new and useful term is found, or where a knowledge base problem is detected.

Feedback potential: Low. The technique itself does not result in significant feedback, but feedback will normally result from interrupting it to structure concepts.

Group-efficacy: Suited to individual use, although the process of deciding how to deal with discrepancies can be effectively done with several experts involved.

Our experiences: We have had considerable success manually scanning documents to ensure that their terms and concepts correspond to those in the knowledge base; editing the document or knowledge base where necessary. So far we have used the computerised version of this technique only on small exapmples, however there has been considerable interest in it in connection with our ongoing BNR project.

Example: We took a software specification document and carefully scanned it. We found numerous subtle problems in *almost every* sentence. The process of correcting the knowledge base and the document was edifying to all concerned. Prior to this process, the document was considered 'correct' to the author and other experts, and the experts were slightly hostile to our attempts at critiquing. After the knowledge acquisition process (a large portion of which was simply discussion of the detected problems), the experts became very positive towards our technology, as they realized the hidden problems in their document.

- ***Conceptual diagram drawing***

Starting point: Several existing types and properties (relations)..

Result: An arbitrary graph of concepts: typically a mix of new concepts and refined concepts.

Technique: The user manipulates a graph composed of nodes and links (see figure 3). The process normally starts with a few nodes (usually representing type concepts or instance concepts) on display. The user can add, delete and move nodes around by direct mouse manipulation; the user can also indicate conceptual relations by linking nodes together. Other useful features include the ability to rapidly alter the set of nodes and links that are displayed in order to work on different parts of the knowledge base.

A number of knowledge base tools support this technique, e.g. KSSn (Shaw and Gaines 1991). In CODE4 we have added considerable sophistication for the experienced user, for example the graphing capability is tightly integrated with the outlining ability: The user can create networks of browsers such that concepts displayed in one window are functionally dependent on the user's selection of concepts in a prior 'driver' window. Updates in one window are automatically reflected in others, which may show the concepts in a different context.

Interface modality: Almost completely non-modal.

Interruptability: Completely interruptable.

Feedback Potential: High. Typically the user will have several windows open on the knowledge base, and as changes are made in one window, the consequences can be seen in another window. CODE4 also provides a separate feedback window by which the system can comment textually on what has been entered.

Group-efficacy: Generally an individual technique.

Our experiences: We use diagram drawing extensively, although mostly for display with only minor editing. Most people find conceptual outline structuring (described below) to be more effective when a large number of changes have to be made.

The most popular kind of diagram is the 'isa' hierarchy; second most popular are other kinds of hierarchical diagrams such as categorization of statements, and part decomposition. The ability to create other more complex diagrams is used far less, even though people are impressed when we demonstrate CODE2 and CODE4's capabilities..

Example: When creating a model of a telephone system we used diagram drawing to show how parts are interconnected, and to draw a finite state diagram. All the knowledge contained in these diagrams could also be viewed by looking at properties of concepts using conceptual outline structuring (e.g. a state has in-transition and out-transition properties). This provided useful feedback.

Figure 3: Two windows for conceptual outline structuring (left) and a window for conceptual diagram drawing (right). The knowledge is from our Smalltalk project and is expressed quite formally.

- ***Conceptual outline structuring***

  Starting point: Several existing types and properties.

  Result: As with conceptual diagram drawing, an arbitrary graph of concepts (although only a graph in the mathematical sense, not displayed as such)

  Technique: As with conceptual diagram drawing, the user uses direct manipulation to manipulate nodes and links. However the nodes are lines of text, and the relations between the nodes are indicated by indentation. This technique is largely a textual analog of conceptual diagram drawing, with an appearance somewhat like a conventional outline processor (see figure 3). In order to facilitate learning, the user inputs required in CODE4 to manipulate nodes and links have been made almost identical to those of the conceptual diagram drawing technique

  Interface modality: Almost completely non-modal.

  Interruptability: Completely interruptable.

  Feedback Potential: High, in the same way as conceptual diagram drawing.

  Group-efficacy: Generally an individual technique.

  Our experiences: This technique is the most heavily used in the CODE2 and CODE4 repertoire. People use it to structure 'isa' hierarchies and the statement hierarchies that describe an individual concept.

  The main differences between the diagraming and outlining techniques (aside from visual appearance) are the following:

  1) Much more knowledge can be displayed in a window using outlining. This is both because of the reduction in white space, and because nodes can be physically larger, containing useful secondary information (this cannot be done in a diagram because it would prevent effective screen layout).

  2) Impure hierarchies (e.g. those involving multiple inheritance) and arbitrary directed graphs are much more easy to manipulate using diagram drawing because it is two-dimensional

  We find that people's preferred mode of thinking or SET (Meyer and Ross 1991) (visual or textual/aural) tends to influence choice of technique. This is why we claim that these techniques should be considered distinct, despite the fact that the users interactions are very similar: People definitely feel that they are using different techniques.

- ***Property table completion***

  Starting point: A largely complete conceptual structure.

Figure 4: A window for property table completion from the Cogniterm knowledge base. Note that most of the knowledge is very informal natural language.

Result: A refinement of the conceptual structure to fill in gaps in the knowledge.

Technique: Concepts are are listed on one axis, and properties are displayed on the other axis of a table (see figure 4). The values of statements (whose subject is a concept and whose predicate is a property) are displayed in cells of the table. The user can edit the table, particularly the values of the cells, somewhat like a spreadsheet.

Interface modality: Largely non-modal.

Interruptability: Completely interruptable.

Feedback Potential: Very high. Editing affects other windows and the feedback panel in the same way as with diagraming and outlining. This technique gives valuable feedback allowing the comparison of concepts, highlighting discrepancies. Numerous different tables can be dynamically generated by selecting appropriate sets of concepts and properties.

Group-efficacy: Generally an individual technique.

Our experiences: This has been a relatively new addition to our CODE2 repertoire and has been largely used to display tables with little editing. Users have found it prevents 'flipping' between concepts or opening large numbers of windows, and makes the process of comparing concepts much easier. Further research is needed, particularly the addition of the technique to CODE4.

- ***Systematic statement critiquing***

Starting point: A moderately complete knowledge base.

Result: A more complete knowledge base with 'problems' removed.

Technique: Statements in the knowledge base are presented to the participants for critiquing in a structured order. They are normally syntactically embellished to look like natural language. Statements may be immediately changed if problems are found, or the problems may be recorded so they can be fixed by individual experts. The choice of immediate or deferred updating substantially effects the sequencing of statements that should be presented.

We find it best to present a blocks of statements (e.g. a related subset of all those about a concept) on the screen and ask the participants to read them and speak up if they detect any problems. It is particularly important to be systematic in the order of presenting the statements, e.g. starting from the most general concepts (once the most general concepts are found to be correct, there are less likely to be problems with inherited knowledge).

We envisage expanding this technique when it is done in a group so it operates like a software code inspection; i.e. with paraphrasing and the discipline of pre-examining the knowledge base by participants.

When critiquing participants should ask the following questions of each statement: Is it: 1) Correct with respect to the opinion of the experts? 2) Consistent within the knowledge base? 3) Unambiguous in its expression? 4) Sufficiently formal to be useful? 5)

Represented in the most appropriate way? 6) Sufficiently general or specific? 7) Sufficiently complete? and 8) Relevant to the purpose of the knowledge base?

Interface modality: Largely modal.

Interruptability: Interruptable, but it is best if interruption is not done excessively, as the idea is to be systematic in covering the knowledge base. If the user jumps out of sequence and edits knowledge elsewhere, it may be necessary to restart the process as previously validated knowledge may no longer be valid. Of course, correcting knowledge may have a similar effect: An adequately intelligent system would take this into account when sequencing the statements to be presented.

Feedback Potential: Moderate. The knowledge is presented, and any corrections may cause other displays to update, reflecting consequences. But as was noted under 'interruptability' it is best to focus on the current statement.

Group-efficacy: Typically performed individually; however 'inspection' of the knowledge base by several people in parallel can be useful for detecting and correcting problems that arise out of differences of opinion (i.e. consensus-building)

Our experiences: At the current time, we use the user interface of the outlining technique, and rely on the user's familiarity with the knowledge base to systematically present statements to the participants for critiquing. With future research we hope to have the system help organize the session.

Example: In the early phases of the BNR project, individual experts would enter knowledge using outlining and then get together for group critiquing sessions led by a knowledge engineer. CODE2 assisted by a) selecting material to be reviewed, and b) recording the status of group decisions about each statement. These sessions proved extremely valuable, although they were sometimes quite arduous as the knowledge was very complex.

- ***Extended inference***

  Starting point: A largely complete knowledge base.

  Result: Further refinement of the knowledge base.

  Technique: This technique has much the same purpose as systematic critiquing: finding problems with the knowledge. The difference is that specialized automatic inference methods are used.

  We assume that there are some basic inference methods that run in almost real-time, computing new knowledge (forward-chaining) or detecting problems (backward chaining). CODE4 has inheritance, delegation, value combination and a number of others. When a knowledge base nears completion, however, we may want to unleash the full power of first order logic to perform inferencing that is too computationally complex to perform in real time. We call this *extended inferencing*.

  The user must request that extended inferencing be run. He or she may request that it be performed on a particular part of the knowledge base and may specify parameters such as search depth. The system then computes either forwards or backwards and reports its results. In the future we envisage the results being presented in such a way that the user can perform systematic critiquing of them.

  Interface modality: Non-modal. The user is presented with a list of consequences with which to do whatever he or she wants.

Interruptability: This technique is inherently atomic from the user interface point of view, so interruption is not possible (of course, the on-going computation could be aborted, perhaps displaying results computed so far).

Feedback potential: Very high. Feedback is the *raison d'être* of this technique.

Group-efficacy: Not applicable. It only takes one user to start the inferencing (although subsequently a group may use the critiquing technique on the consequences).

Our experiences: We developed a Prolog-based adjunct to CODE2 called FOLDE (Full First-Order Logic Deduction Engine) (Skuce 1991). Interestingly, we found nobody but ourselves was interested in using it, and we have not yet used it in any of the major projects listed in section 1.2. We believe that this is partly due to the following: a) the need to fully formalize any input knowledge in order to use extended inferencing; b) the fact that its user interface is not yet adequately developed; and c) lack of familiarity with, and fear of, logic. Further research is needed.

Table 1 summarizes and quantifies some of the analysis of techniques given in this section. All of the attributes are rated on a scale of 0 to 10; the values assigned are the authors' judgements, and should primarily be considered relative to each other.

Technique	Average formality of starting point	Amount of new knowledge (vs. refined)	Average formality of result	Potential for creativity	Level of feedback provided	Group efficacy
Brainstorming	0	10	1	10	7	10
ISM	2	4	4	3	3	9
Scanning	1	8	6	1	1	3
Diagraming	5	6	8	9	8	4
Outlining	4	8	7	9	9	5
Table completion	7	3	8	5	10	5
Critiquing	6	1	9	3	5	6
Extended inference	10	2	10	2	10	6

Table 1: Ratings of some of the eight techniques of figure 1 in terms of the questions posed in section 2.1

Table 2 lists the knowledge engineering projects we described in section 1.2, and indicates the degree to which we used each technique. It also ranks some of the projects in terms of the questions asked in section 2.1. The columns have the following meanings:

X: Relative complexity of the knowledge bases produced, on a scale of 1-10

F: Formality of the resulting knowledge base, expressed as a percent.

N: Percent of time spent adding knowledge as opposed to refining or debugging it

C: Percent of the resultant knowledge base that was creative, (i.e. that contained knowledge that was not present anywhere in any form prior to the project)

G: Percent of work performed in groups

As with table 1, the figures in table 2 are the authors' judgements and should be considered in a relative sense. In future research we expect to instrument CODE4 so that some of these figures can be measured.

Project	X	F	N	C	G	Techniques Used	Comments
Telos	10	15	30	40	50	35% Outlining	In the early and late phases, experts did most of this. In the intermediate phase most was done by a knowledge engineer after reading documents and interviewing experts.
						55% Critiquing	In the early phases, this technique was performed intensively. During the time when the knowledge engineer was performing the outline browsing, the statement critiquing became easier.
						5% Diagraming	At the time, CODE lacked the diagram-drawing power it now has
						5% Scanning	Some design documents were scanned and the documentation group became interested in this technique to make sure that all terms were properly described in the knowledge base
Telebrain	6	30	70	90	5	65% Outlining	The knowledge engineer and expert were the same person, resulting in
						15% Diagraming	It became particularly useful to represent engineering diagrams directly in the knowledge base.
						20% Critiquing	This figure was lower than normal because this project was almost entirely inventive in nature, this there was little checking for correctness, only consistency, completeness, ambiguity etc.
Ontology	2	25	10	50	10	30% Outlining	
						10% Diagraming	
						60% Critiquing	The knowledge bases created for this project are typically very small, so the focus is on maximizing the correctness and utility of the statements
Self-description	4	15	45	60	30	5% Brainstorming	
						5% ISM	
						5% Scanning	During several design efforts, natural language documents were done first
						25% Outlining	
						10% Diagraming	
						30% Critiquing	
Cogniterm	7	10	65	10	20	30% Outlining	
						20% Diagraming	
						10% Table completion	This technique is particularly important when comparing concepts or writing definitions.
						30% Critiquing	
Smalltalk	4	35	60	20	5	70% Outlining	As with Telebrain, the knowledge engineer and expert were the same person
						30% Critiquing	

Table 2: How we applied the techniques to the projects described in section 1.2. See the text for a description of the lettered columns.

2.3 Other techniques

It is important to understand that we are not advocating the abandoning of many other tried and true techniques, such as those surveyed by Neale (Neale 1989). We have just not yet found it useful to integrate semi-automated support for them into our environment. We add techniques when there is sufficient demand from users, or when we have sufficient evidence that an experimental implementation would be worthwhile.

For example, there are many techniques in the literature to help a knowledge engineer conduct interviews. We try to encourage experts to work directly with our system by providing a simple and flexible user interface, and permitting very informal input. However, where interviews are necessary we see little advantage in computer assistance: A skilled interviewer can do an effective job of elicitation and paper-and-pencil recording of results. The computer integrated techniques come into play following interviews when the knowledge engineer typically uses outline structuring to enter the elicited knowledge.

Repertory grids are used in another popular technique (Gaines and Shaw 1987). We have, however, found only minimal use for them in the kind of knowledge engineering projects we do because of their limited expressiveness. The closest we come is our table completion technique, which is more general in one sense, but which does not provide the same kind of dynamic feedback. Interestingly, the knowledge gathered for the tables in this paper may have benefited from repertory grid analysis.

2.4 Combining the computer-supportable techniques into a methodology

Different knowledge engineering project require different techniques. As we pointed out in section 1.1, we focus on descriptive modeling projects, but even within this constraint, projects can require different techniques.

It is typical for 'methodologies' for various activities to prescribe a relatively fixed procedure to achieve some goal. We believe, however, that just about any technique can have its place and it is up to the skill of the knowledge engineer to pick the right technique and to switch back and forth between techniques as needs dictate. In our view, therefore, a good methodology should be no more than a set of heuristics that suggest when a given technique can be useful.

The most important heuristic we espouse is 'use the technique that appears the most useful at the time, and feel free to switch among techniques'. However, our analysis of the above techniques (including the analysis presented in the tables) has led us to develop a number of heuristics. The heuristics are still under development but the following issues are of major importance:

Role of knowledge engineer

- Skilled knowledge engineers tend to do most of their work using conceptual outline structuring. We encourage non-knowledge-engineers to directly use the system: They tend to also use outlining extensively, although with fewer of its powerful features. They also tend to want to see knowledge represented in forms such as diagrams and tables.

Sources of existing knowledge

• If the knowledge is largely in people's minds then brainstorming, ISM and conceptual structuring are effective at obtaining it. Quite often though, there is a substantial body of written material. We find it important to scan this written material, both to ensure its content is captured and to ensure that it is not inconsistent with the knowledge base. The latter problem is extremely common.

Complexity and amount of knowledge

• Knowledge bases can get very large and complex, and the classic 'knowledge acquisition bottleneck' can hinder further progress. With astute use of partitioning aids and feedback mechanisms in the conceptual structuring techniques it is possible to 'hide' much of the complexity. Systematic critiquing can prevent excessive iteration when trying to debug a complex knowledge base.

Phase of the project

• Typically the early phases of a project will involve more use of brainstorming and other techniques that are best at generating new knowledge. In later phases, the refining techniques are used more. We find it useful, though, to expand and refine the knowledge base iteratively, focusing on somewhat different areas each time. This idea has been explored by Shaw and Gaines (Shaw and Gaines 1991)

Intended use of the knowledge

• If the knowledge acquisition is being largely performed to help understand a domain, rather than for the development of an inference-performing system (e.g. the typical expert system), then less formality is needed in the resulting knowledge. This does not, however, mean that critiquing is not necessary; it means that the system need not 'understand' the representation of every concept.

3. INTEGRATION

We have used the following principles when integrating the techniques into CODE4:

1. All of the techniques operate in windows and act on an underlying knowledge base using a well-defined series of internal editing commands (i.e. there is a common API between all the user interface windows and the knowledge engine).

2. We have defined a higher-level API for techniques that involve displaying or traversing sets of related concepts (diagram drawing, outline processing, property table completion and systematic statement critiquing). This API has its own set of common internal commands operating on what we call 'knowledge maps' that each describe a particular view of a knowledge base.

3. The user interfaces for all the techniques have a simiiar look and feel.

4. Changes made in one window are, by default, reflected in others immediately

5. Windows for different techniques can be made to 'drive' one another with selections made in one window determining what is displayed in another window on a different technique.

4. SUMMARY AND CONCLUSIONS

In this paper, we have presented a series of knowledge acquisition techniques for which we are developing integrated computer support within CODE4. Some of these (e.g. Interpretive Structural Modelling, conceptual outline structuring) have not been extensively discussed in the knowledge acquisition literature and we therefore desire to point out their advantages.

The techniques vary in a number of dimensions. Some, such as brainstorming, and diagram- or outline-oriented conceptual structuring stimulate creativity. Some, such as table completion, extended inference and conceptual structuring give the users a large degree of feedback, helping them to understand the consequences of the entered knowledge. Brainstorming and ISM are the techniques most suited to use in a group, although we have had considerable success with group-oriented critiquing. Brainstorming, conceptual structuring and scanning are most effective at generating large quantities of new knowledge; while critiquing, extended inference and table completion are the most useful for refining knowledge.

Conceptual outline structuring and critiquing are the techniques that are used most often. Technique usage depends strongly on factors such as the type and complexity of the project and the knowledge engineering skill and personal preferences of the users. We find that a large, complex project tends to involve much more use of the refinement-oriented techniques. A skilled knowledge engineer uses mostly conceptual outline structuring, although conceptual diagram drawing is used by visually-oriented thinkers and in cases where non-hierarchical relationships are important.

ACKNOWLEDGEMENTS Support for this research has been provided by Bell-Northern Research and the Natural Sciences and Engineering Research Council of Canada.We would like to thank the following people for their work on various knowledge engineering projects: Ingrid Meyer, Karen Eck and Lynn Bowker for Cogniterm; Yves Beauvillé and Joe Sarkic for the self-description project; Ken Iisaka for the Smalltalk project, and numerous people at BNR for the Telos project, especially Roy MacLean and Milan Javor.

REFERENCES

Bradshaw, J., C. Chapman, et al. (1992). "An Application of DDucks to Bone-Marrow Transplant Patient Support". *1992 AAAI Spring Symposium on Artificial Intelligence in Medicine*, Stanford University, March.

Bradshaw, J., P. Holm, et al. (1992). "eQuality: A Knowledge Acquisition Tool for Process Management". *FLAIRS 92*, Fort Lauderdale, Florida.

Delbecq, A. L. and e. al (1975). *Group Techniques for Program Planning*. Glenview, IL, Scott, Foresman & Co.

Gaines, B. and M. Shaw (1987) *Knowledge Support Systems*, MCC.

Iisaka, K. (1992) *Knowledge Base of Fundamental Concepts of the Smalltalk-80 Language and Programming Environment*, Undergraduate Senior Report, University of Ottawa.

Lenat, D. and R. Guha (1990). *Building Large Knowledge Based Systems*. Reading, MA, Addison Wesley.

Lethbridge, T. C. (1991a). "Creative Knowledge Acquisition: An Analysis". *6th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff.

Lethbridge, T. C. (1991b). "A model for informality in knowledge representation and acquisition.". *Workshop on Informal Computing,* Santa Cruz, Incremental Systems.

Lethbridge, T. C. and D. Skuce (1992). "Informality in Knowledge Exchange". *AAAI Workshop on Knowledge Representation Aspects of Knowledge Acquisition*, San Jose.

Meyer, I., D. Skuce, et al. (1992). "Towards a New Generation of Terminological Resources: An Experiment in Building a Terminological KB". *13th International Conference on Computational Linguistics (COLING).*, Nantes.

Meyer, M. and J. Ross (1991). "Psychotherapeutic techniques for identifying experts' and system users' sensorial experience of thinking: Significance to knowledge acqusition". *proc. 6th Banff Knowledge Acqusiition for Knowledge-Based Systems Workshop*, Banff.

Miller, G. (1990). “WordNet: an On-line Lexical Database.” *International Journal of Lexicography.* 3(4): whole issue.

Neale, I. M. (1989). “First generation expert systems: a review of knowledge acquisition methodologies.” *The Knowledge Engineering Review* : 105-145.

Saunders, C. (1990) *A PRISM Handbook*, , BNR, Internal Document.

Shaw, M. and B. Gaines (1991). "Using Knowledge Acquisition Tools to Support Creative Processes". *proc 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff.

Skuce, D. (1991). “Interactive Debugging of First-order Knowledge Bases.” *Data and Knowledge Engineering* submitted.

Skuce, D. (1992a). "Knowledge Management in Software Engineering: A Tool and and Experiment". *7th Knowledge Based Software Engineering Conf.*, Tyson's Corner, VA.

Skuce, D. (1992b). “Managing Software Design Knowledge: A Tool and an Experiment.” *Resubmitted with reviewers changes to: IEEE Transactions on Knowledge and Data Engineering* : 36.

Skuce, D. (1992c). “A Wide Spectrum Knowledge Management System.” *To appear in: Knowledge Acquisition* : 49 pp.

Skuce, D. and T. C. Lethbridge (1992). "A Knowledge Representation for Interactive Knowledge Management". *Submitted to: 3rd International Conf. on the Principles of Knowledge Rep. and Reasoning*, Cambridge, Mass., October.

Skuce, D. and I. Monarch (1990). "Ontological Issues in Knowledge Base Design: Some Problems and Suggestions.". *Proc. 5th Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff.

Szpakowicz, S. (1990). “Semi-automatic acquistion of conceptual structure from technical texts.” *Int. J. Man-Machine Studies* 33 385-397.

Warfield, J. N. (1976). *Societal Systems: Planning, Policy and Complexity*. New York, Wiley.