Submitted for review (1997) to *Int. J. Human Computer Studies*

Evaluating a Domain-Specialist Oriented Knowledge Management System

Timothy C. Lethbridge

School of Information Technology and Engineering
150 Louis Pasteur, University of Ottawa
Ottawa, Canada, K1N 6N5
tcl@site.uottawa.ca

Abstract

We discuss the evaluation of a tool designed to allow domain specialists to manage their own knowledge base. We present the evaluation as a two-phase process: In the first phase we assess whether the tool has met its objectives of allowing those not trained in logical formalisms to effectively represent and manipulate knowledge in a computer. By studying use of the tool by its intended users, we conclude that it has met this objective. In the second phase of the evaluation, we assess what aspects of the tool have in fact led to its success. To do this we study what tasks are performed by users, and what features of both knowledge representation and user interface are exercised. We find that features for manipulating the inheritance hierarchy and naming concepts are considered the most valuable. Our overall conclusion is that tool research must involve this two-phase approach if the others are to learn from the work – the research has much less value unless it can be determined which features should most profitably be adopted by others.

1. Introduction

This paper presents a case study in evaluation of a domain-specialist oriented knowledge management system called CODE4 (Lethbridge, 1994; Skuce and Lethbridge, 1995). CODE4 is primarily intended to be used by domain specialists who want to organize and manipulate knowledge for the sake of better understanding and communicating that knowledge among humans. Contrasting systems rely on knowledge engineers, and involve the development of formalized models of knowledge for manipulation by software that performs reasoning.

In this paper we outline key steps in the process of developing CODE4, but the focus is placed on evaluation. Evaluation is a key phase in any software development project: In conventional software engineering, evaluation comprises such activities as validating requirement or prototypes to ensure they meet customer needs, and verifying that the final product is free of defects. The star model of software development (Hix and Hartson, 1993) places evaluation at the centre, with all other software development tasks surrounding it.

The hoped-for conclusion from such evaluation is that the system in question is in some sense *good*. Such conclusions are of little value, however, if others are to learn from the software development experience. If software is being developed as a research task therefore, it is necessary to go one step further and evaluate *why* the software is good. In this paper we present both types of evaluation.

It is hoped that readers of this paper will learn two things: The first is what kinds of features ought to be present in a domain-specialist oriented knowledge management system

– as determined by our evaluation. The second is how should research regarding software tools be performed and reported – we hope that this paper highlights some important aspects of that process.

1.1 A general approach to software tool research

This paper is structured according to the following general approach to performing research into software tools. The intent is that these steps be performed iteratively:

i. Determine the task(s) to be performed and the target population(s) of users.

ii Establish evaluation criteria that will be used to determine the success of the to-be-built tool.

iii. Design the tool.

iv. Perform effectiveness evaluation: i.e. evaluate the tool to determine whether success has been achieved according to the criteria established in step ii.

v. Perform analytic evaluation: i.e. evaluate features of the tool to determine which ones contributed to the success.

It is our belief that each of these steps is important, but that much research into software tools does not effectively address one or more of the steps.

Step i is often omitted because the tool is developed to test or implement some underlying technological of scientific idea (e.g. a new knowledge representation), rather than to help perform a task. Unless the target task and users are effectively chosen, few or no conclusions about usefulness can be drawn. On the other hand, if a real-world need can be established by defining a task that users want to perform more effectively, then the resulting system can be evaluated to determine if the users are in fact more productive. Task analysis is well understood in the field of human computer interaction (see Diaper, 1989), but is not widely applied by researchers in computing disciplines

Step ii is also often omitted when software tools are developed in a research setting – the evaluation criteria are often established only during the evaluation phase, after the tool exists. However, establishing evaluation criteria can help focus the work and guide design decisions. Establishment of evaluation criteria in advance is characteristic of engineering approaches such as usability engineering (Whiteside, Bennett and Holtzblatt, 1988).

In the past there has been criticism that software developed in research environments, and reported in the literature, is inadequately evaluated. Although we are now in the era where the importance of evaluation is more widely recognized, we believe that little attention is placed on distinguishing steps iv and v in our general approach. If step iv is performed alone, readers are able to conclude that the researchers have done good work, but they cannot learn what aspects of the tool contributed to success. Readers might assume that all of the features described were useful – in fact, this is the impression conveyed by much literature that describes software tools. On the other hand, if detailed evaluations of features are performed (step v) but there is no verification that the tool as a whole is useful (step iv), then the reader is equally unable to obtain insights that can be reliably applied to subsequent work.

This paper, therefore, is structured according to the above general approach. Section 2 discusses the first three steps, presenting the task for which CODE4 was intended, as well as key aspects of its design. Section 3 discusses CODE4's evaluation in general, focusing on the overall techniques used; section 4 presents the effectiveness evaluation and section 5 presents the analytic evaluation.

2 The task and the design

In this section we summarize those aspects of our research that preceded evaluation: steps i, ii and iii as described in section 2.1

2.1 Task analysis and the target population

The first step in our research was to define the task. Table 1 presents two very different tasks that might be performed under the general heading of 'knowledge management'. The first task is by far the most widely researched: Knowledge engineers develop models of some domain, and represent the knowledge using languages rooted in formal logic. The knowledge is then used to drive formal reasoning processes in a computer program commonly called a 'knowledge based system'.

The second task in table 1 is not so widely researched: Domain specialists, typically with no intent to create a knowledge based system (i.e. no intent to develop a system that performs automated reasoning), want to manipulate knowledge structures so as to understand better some aspects of the world. Analogous tasks would be financial analysts who want to manipulate numbers in a spreadsheet, to better visualize the data and to discover useful relationships using the analytic capabilities of the human brain. Lacking CODE4, domain specialists typically structure their knowledge using diagrams, written text, spreadsheets, outline processors etc. Our hypothesis in performing this research was that a simple tool for manipulating concepts and properties (frames and slots) would prove very useful to these people.

Both task 1 and task 2 involve the acquisition of knowledge, and will probably require a system that can represent such entities as concepts and relations. However, the nature of the tasks will lead to design decisions and evaluation criteria that are significantly different.

Why did we choose to develop a system for task 2 instead of task 1? The main reasons were that we saw an unserviced market and a lack of research into the task.

2.2 Criteria for evaluation of effectiveness

The success of tools and languages developed with task 1 (in table 1) in mind can probably best be judged by evaluating whether the inferences made with the resulting knowledge are sound and allow people to make decisions effectively. These criteria can generally be specified in objectively-evaluable terms – for example verifying that a tool comes to expected conclusions in known cases, and performs at least as well as other tools.

For our task-2 research, however, any inferences made are by humans. We are therefore forced to use more subjective evaluation techniques.

We decided that the success would be judged based on two criteria. The criteria are phrased as questions and described below:

Criterion 1: Is the tool used on a discretionary basis by significant numbers of members of its target audience and on a variety of different and significant examples of the intended task?

By using this criterion we are asserting that if the system is used voluntarily by people, then there must be value in it, and our hypothesis mentioned above is found to be true. The criterion requires that the usage be by its target audience and on a variety of significant examples so that we can gain confidence that we are not drawing conclusions by chance.

	Task 1	Task 2 (our research)
Description of task	Developing formalized models for manipulation by software that performs reasoning.	Organizing and manipulating knowledge for the sake of better understanding or communication of that knowledge among humans.
Developers of the knowledge base	Knowledge engineers	Domain specialists
Typical knowledge base development subtasks	Eliciting knowledge; encoding it in a formal representation language; validating it.	Modeling one's personal understanding of the domain at whatever level of formality proves useful; validating the knowledge.
Agents that perform reasoning with the knowledge	Computer programs	Humans
Typical tasks that involve use of the knowledge base	Entering parameters; running the program to obtain inferences; analysing the results to make decisions.	Massaging the knowledge to obtain different views of it and detect patterns; making decisions based on a better understanding of the domain.
Key evaluation criteria used to determine if the system is successful	a) Are inferences that are made using the knowledge sound? b) Using the resulting knowledge and inferences, can decisions be made at least as easily as they could using other methods?	a) Is the system used on a discretionary basis on significant examples of the task? b) Is the system perceived to be at least as useful as other techniques that could be used for the task?
Key design issues (most important listed first)	Choice of knowledge representation(s) that will allow effective encoding of the knowledge and allow inference methods to work effectively. Choice of knowledge acquisition method(s). Choice of reasoning method(s).	Choice of mediating representation(s) that will allow humans to better understand the knowledge. Choice of knowledge manipulation method(s). Choice of knowledge representation(s) that will allow effective encoding of the knowledge and work with the mediating representations effectively.

Table 1: Comparison of two very different knowledge management tasks

Criterion 2: Is the system perceived to be useful, i.e. at least as useful as other techniques that could be used for the task?

The second criterion provides additional evidence to back up the first. Although it is unlikely that people would use a non-useful system voluntarily, they might choose a system that is marginally useful, and they might choose a system without being aware of other systems that could do the job better. The question posed by this second criterion can be answered by asking users about, a) what tasks they were able to perform with the tool, and b) how their experiences using this tool compared with experiences using other tools.

Our objectives were to obtain strong positive answers to both of the above questions. The conclusions are presented in section 4.

2.3 Design decisions based on the task and evaluation criteria

This section outlines some key decisions made during the design of CODE4, and explains their rationale in terms of the task presented in section 2.1 and the evaluation criteria presented in section 2.2. More complete details can be found in Lethbridge 1994 – only enough is presented here so the reader can understand section 5 which discusses analytic evaluation.

To achieve our objectives, we had to design two elements: the knowledge representation and the user interface.

We had to design a knowledge representation that had sufficient power to represent the kind of information users wanted to represent, yet would not require much training to get started. In particular we did not want users to have to put up with complex textual syntaxes; nor did we want them to have to know formal logic or computer programming.

The user interface had to be able to effectively present different views of the knowledge to the users. It also had to allow them to edit knowledge without having to follow restrictive syntactic conventions.

Each of the above aspects is summarized below.

Overview of the knowledge representation

We attempted to keep the knowledge representation very simple: For the sake of uniformity, everything the users manipulate is a concept, and all concepts have common features – e.g. they are located in an inheritance hierarchy and they inherit properties (i.e. relations). Since everything is a concept, even the properties are concepts.

Concepts are divided into type concepts, that can have subconcepts (specializations) and instance concepts that represent specific things. In order to allow the user to more easily manipulate concepts, he or she can readily convert a concept between instance and type (and vice versa if it does not have any subconcepts).

In order to ensure that users could work with the knowledge without being constrained by syntax, we introduced several features that allow what we call 'informal' knowledge to be manipulated: Concepts can have several different terms (synonyms), and the values of properties can be expressed using arbitrary text. Users can add more formality if they choose by making the values of concepts point to other concepts. See Lethbridge and Skuce (1992) and (1994) for more on this topic.

Several features were added in order to allow users to better visualize the knowledge. One such feature was the global property hierarchy – this contains all properties in the system, with each concept inheriting a subhierarchy of it. Other features added for visualization purposes included graphical icons and graph layout information.

In order to represent information such as multiple terms, property hierarchy relations, icons and graph layout data without adding special features (i.e. without veering from the notion that the knowledge base should consist entirely of concepts), the notion of the 'metaconcept' was introduced. Metaconcepts are not intended to be used directly by ordinary domain experts – they serve only as special concepts that contain information *describing* an underlying concept as opposed to information described *by* the underlying concept. Each concept, conceptually at least, has a metaconcept; information about a concept is described using properties of its metaconcept that are known as facets.

A relatively small number of other features were added to the representation to solve specific problems encountered by groups of users. Among these were a) a 'combining' technique for automatically resolving situations where two different values of a property are inherited due to multiple inheritance; b) an ability, called 'delegation', to specify the value of a property by reference to another property; c) an ability to specify that two type concepts are disjoint (i.e. cannot have a common descendent in the inheritance hierarchy); and d) an ability to add 'dimension' or discriminator labels to distinguish among alternative ways of dividing a concepts into subconcepts. See Bowker and Lethbridge (1994a) and (1994b) for information on the latter.

Overview of the user interface

The user interface was designed so that users can interact with multiple different views of the knowledge. Four so-called mediating representations were developed; i.e. visual representations that mediate between the user and the underlying knowledge representation.

Subwindows, each containing a view of part of the knowledge base, are chained together such that sets of concepts currently selected by the user in one window dynamically determine what appears in dependent windows. Each subwindow uses one of the four mediating representations and shows concepts linked by one or more properties.

The four mediating representations are as follows:

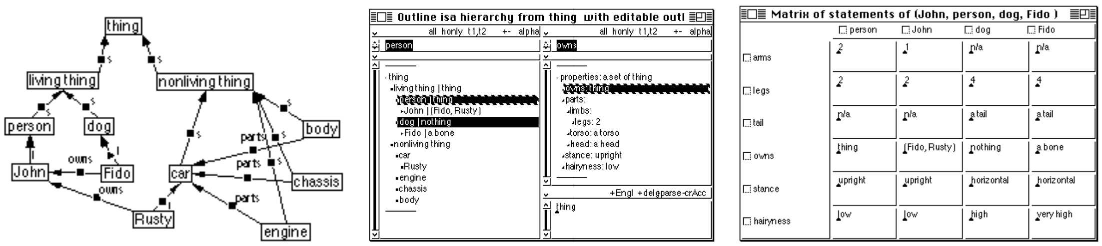
1. Graphical: This lays out concepts as nodes linked by arcs in two-dimensional space;. Users can edit the graph by adding nodes and arcs; they can also manually adjust and save the layout of the graph.

2. Outline: This lays out concepts as in the outline view of a word processor, with indentation representing the relation of one concept to another. Concepts can also be arranged alphabetically.

3. Matrix: This lays out concepts in rows and columns with the relations between concepts shown as the intersections of the rows and columns.

4. User language: This is a degenerate mediating representation that merely shows the value of a property as text.

Figure 1 gives capsule views of windows containing the various mediating representations.



a) b) c)

Figure 1: The CODE4 mediating representations showing concepts and relations. a) graphical; b) two outline views and a user language view at bottom right; c) matrix

The first three mediating representations have many features in common. In particular, the user can: a) add and delete concepts and relations; b) select sets of concepts in several standard ways; c) cause a subset of concepts to be shown or highlighted according to the setting of a ‘mask’.

In addition to the above, there is a control panel facility that allows the user to specify general preferences as well as load and save knowledge bases.

In order to modify knowledge, users open a knowledge base (or request to start a new one). They then edit the knowledge using one or more of the mediating representations. In order to query and explore the knowledge they can do one of the following: a) Use the masking facilities to pick knowledge of interest; and, b) open subwindows that show concepts and relations of interest. When used together, these facilities give the user quite a rich knowledge management environment.

3. General aspects of the evaluation

In order to provide a basis for both the effectiveness and the analytic evaluation (sections 4 and 5) we performed a circumscribed study of a set of 12 users as they developed 25 knowledge bases. This section describes that process.

3.1 The user study

The basic steps in performing the user study were: 1) Solicit people to build knowledge bases; 2) Train them; 3) Perform any necessary enhancements to the system in order to accommodate their needs, and 4) Study the results of their knowledge base building efforts. Details of these steps follow:

Step 1: As many people as possible were solicited to build knowledge bases using CODE4.

Some users were solicited from the University of Ottawa student or researcher population – people who had to perform tasks for which CODE4 was suitable. Users from companies and other universities also started using the tool after they found out about the product through various publications and contacts. Section 4 discusses tool usage in more detail.

Step 2: The users were trained to use the system, or trained themselves.

A 100-page user manual was created to train users. Additionally, users received training: 1) from graduate lectures, 2) by following a step-by-step tutorial created by members of a particular user group, and 3) by personal consultation with members of the development team.

Step 3: CODE4 was regularly enhanced in response to requests from the users.

It was not possible to have CODE4 remain static during the research. Significant new features were added over a three-year period in order to meet the requirements of users. The kinds of features added after CODE4 started to be used seriously include:

- Knowledge representation features such as dimensions.
- New mask predicates to permit specialized queries.
- New primitive properties to permit the display and storage of specialized information such as multiple graph layouts
- New display options for existing mediating representations.
- An entirely new mediating representation: the matrix.

Step 4: Information was gathered from as many users as possible

A significant percentage of CODE4 users were willing to have their work studied in detail. These users are called the *participants*[1] in the following discussions.

The total number of participants was twelve; all used CODE4 at the University of Ottawa. Eleven were graduate students and one was a professor. Student usage was divided between those using CODE4 for course work, and those using it in thesis research.

The participants created 25 knowledge bases. These covered a very wide range of topics, categorized as follows:

- Computer languages and operating systems (eleven knowledge bases).
- Other technical topics (e.g. optical storage, electrical devices, geology, matrices – nine knowledge bases).
- General purpose knowledge (e.g. people, vehicles, fruit, top level ontology – five knowledge bases).

The total amount of work involved in creating these knowledge bases was reported to be about 2000 hours, i.e. an entire person-year.

The work of the participants was studied in two ways:

1) By analysing the participants' responses to a detailed questionnaire and follow-up interview questions. Further details about the questionnaire are found in the next sub-section.

2) By measuring the knowledge bases. Some of the metrics are discussed in section 5. Further details can be found in Lethbridge (1998).

3.2 The CODE4 user questionnaire

The questionnaire contains six parts containing a total of 55 main questions, many of which have sub-questions: There are a total of 259 sub-questions about CODE4 in general and

[1] There were other users in *addition* to the participants. These did not actively participate (filling out questionnaires, being interviewed etc.) because they were busy, their work was confidential etc.

130 sub-questions about each knowledge base. Most participants took several hours to complete the questionnaire and some were quite enthusiastic about it despite the amount of work involved.

The first two sections of the questionnaire ask general questions in order to ascertain how much users know about CODE4, how they learned it and how much they have used it. The subsequent two sections focus on the users' reactions to specific features of the software. The fifth section deals with users' experiences developing individual knowledge bases. Questions in this section were designed to complement the measurement of the knowledge bases. The final section of the questionnaire covers problems and desires users encountered in the use of CODE4.

Data resulting from the questionnaire is presented in sections 4 and 5.

3.4 Why not run a 'controlled' experiment?

The evaluation methodology described above is that of a *natural* experiment (i.e. trying to reach conclusions about an activity by observing as many cases of that activity as possible, but without interfering). An alternative approach would have been to design a controlled experiment.

In the field of knowledge engineering, a controlled experiment might involve a procedure like the following: First, create two versions of a knowledge management system that differ according to the presence or absence of some feature. Then have separate sets of users create knowledge bases with each system. Finally, measure the resulting knowledge bases to see if the presence or absence of the feature causes any difference in the resulting work.

Such an experiment involves varying one parameter (in this case a feature) while ensuring that differences in other factors, including ongoing changes to the system, do not influence any conclusions. This was judged impractical for the following reasons:

- It is not reasonable to assign a small number of people to create knowledge bases and expect that the resulting work will be comparable. Knowledge enterers have dramatically different levels of skill (analysis ability and understanding of knowledge representation) as well as background (general knowledge that they can apply to the problem).
- It might be possible to control for skill and background differences by using a large number of knowledge enterers. However it was found to be too difficult to find enough people with sufficient motivation to work for tens of hours on a single assigned problem. People were only willing to be participants if they could work on a topic that interested them.
- It might be possible to overcome biases imposed by a particular domain by having participants create several knowledge bases from a small pool of domains. Again however, finding willing people is extremely difficult (it was even difficult to get enough participants, who had already created knowledge bases, to fill in a questionnaire).
- During the conduct of this study, users would frequently ask for new features. If all participants were *assigned* a topic, it might have been possible to deny their requested modifications for the sake of gathering 'good data'. However since most users were working on their own research projects, it was necessary to make regular incremental modifications to the system. Otherwise many users would not have felt comfortable trusting CODE4 to be a reliable tool for their research.

In conclusion, without a very significant amount of money with which to pay participants for long periods of time, it was only feasible to perform an exploratory study of the type described in the previous subsections. We believe, however, that such a study provides adequate information to allow us to effectively evaluate the tool.

4. Effectiveness evaluation: Assessing whether or not success has been achieved

In this section we discuss our assessment about whether CODE4 was successful in terms of whether it met the objectives set for it. The rationale behind the objectives themselves are described in section 2.2.

4.1 Discretionary use on the intended task

The first evaluation criterion, as described in section 2.2, is: *Is the tool used on a discretionary basis by significant numbers of members of its target audience and on a variety of different and significant examples of the intended task?*

A total of at least 30 individuals are known to have used CODE4 to build knowledge bases. These fall into three main groups: 1) industrial customers, 2) researchers and 3) graduate students taking courses. In addition, about ten other groups are known to have obtained copies of a demonstration version. Some examples CODE4's use are described below; all of these involved people voluntarily choosing to use the tool.

The primary industrial customers for CODE4 have been groups headed by Jeff Bradshaw, then of Boeing Aircraft Corporation in Seattle. Bradshaw's work primarily involved building CODE4 into other applications. In Bradshaw, Boose, Shema, Skuce and Lethbridge (1992) an application involving design rationale capture is presented. Another application involving organizational modelling is discussed in Bradshaw, Holm et al. (1992).

CODE4 was also adopted by the Cogniterm project. In this project, linguists used it for terminology research – their goal was to find ways to model multi-lingual terminology to assist translators. Some of the published literature includes Eck's M.Sc. thesis (1993) and papers by Skuce and Meyer (Skuce 1993; Meyer, Eck and Skuce 1994). Bowker (Bowker and Lethbridge 1994a and 1994b) used CODE4 as a major tool for her Ph.D. research.

Several other student research projects and theses have used CODE4 as their major tool; these include Ghali (1993) and Wang (1994). CODE4 has also been used as a teaching tool in several courses in artificial intelligence at the University of Ottawa.

We can conclude from the above that CODE4 has achieved its first objective. The only minor counter-argument is that many of the users completely ignored certain features. However, as stated in section 1.1, we want to evaluate success of the tool as a whole and then subsequently ask question about which features contributed to this success. The latter is discussed in section 5.

4.2 Perceived usefulness

The second criterion to ascertain success is: *Is the system perceived to be useful, i.e. at least as useful as other techniques that could be used for the task?*

To answer this question we posed three subjective questions to the participants of the study described in section 3. The first two questions ask about usefulness in general and the third compares CODE4 to other technologies.

Question 1. How much insight into the *domain* did you obtain?

In almost all instances, the participants report that they obtained substantial insight about the subject matter when using CODE4. On a scale where 0 indicated that constructing the knowledge base resulted in no new insights and 10 indicated that many new insights were obtained, the mean response was 7.1 (range 2 to 10; std. dev. = 2.6).

Question 2. Was creating the knowledge base a worthwhile exercise?

Participants were asked, for each knowledge base, whether creating it was a worthwhile exercise. Answers were on a scale where -5 meant ‘strongly disagree’, 0 meant ‘no opinion’ and 5 meant ‘strongly agree’.

The mean response was 4.2 (range 0 to 5; std. dev. 1.5), indicating a reasonably high feeling that using CODE4 was worthwhile. Users however gave significantly different answers for different knowledge bases: As would be expected, those knowledge bases used for serious research were judged more worthwhile.

Question 3. How easily could you have represented the same knowledge using the following types of software or representation methods?

For each knowledge base they developed, each participant was asked to compare CODE4 with other representational technologies with which he or she was familiar. The answers were on a scale where -5 meant that the knowledge could have been represented much more *easily* using the alternate technology than using CODE4; zero meant that CODE4 equalled the technology in representational ease, and 5 meant that using the alternate technology instead of CODE4 would have resulted in much more *difficulty* (i.e. that CODE4 was much better for the task than the alternative technology would have been be).

In all cases, the mean rating indicates CODE4 is easier for the task than alternative technologies. Table 2 summarizes the results and figure 2 shows a plot of the results. In particular it is notable that CODE4 was perceived as easier to use than both informal representation techniques (e.g. natural language) and formal representation techniques (e.g. predicate calculus and conceptual graphs). Hypertext is CODE4’s only significant competition: One user ranked hypertext as far easier than CODE4, but on average, users still found hypertext a bit harder to use for knowledge management.

Note that the figures only consider cases where participants actually know the competing technology; for example less than half of the respondents compared CODE4 to other AI tools. Also note that the figures do not mean that the competing technologies are less good; the only conclusion that can be drawn is that there is evidence that for *this task*, CODE4 performs better.

Technology	Mean	Max	Min	Std. Dev	Responses (KBs)
Spreadsheet	4.2	5	2	1.2	17
Drawing program	3.9	5	2	1.2	17
Relational database	3.0	5	1	1.6	15
Natural language (word processor)	3.0	5	0	1.7	20
Outline processor	2.9	4	0	1.5	7
Hypertext	1.0	5	-5	2.4	11
Predicate calculus	4.6	5	1	1.1	14
Prolog	4.4	5	1	1.2	11
Other AI tools (any the user had used; e.g. expert system shells)	3.9	5	-1	2.0	8
Conceptual graphs	3.3	5	-1	2.0	12

Table 2: Perceived difficulty of use of representational technologies. larger numbers indicate that the technology is perceived as being more difficult to use for the kind of representational tasks studied. CODE4 (the baseline for comparison) is zero. A negative number would indicate that the technology is perceived as easier than CODE4. There are no negative means; only a few individual negative values.

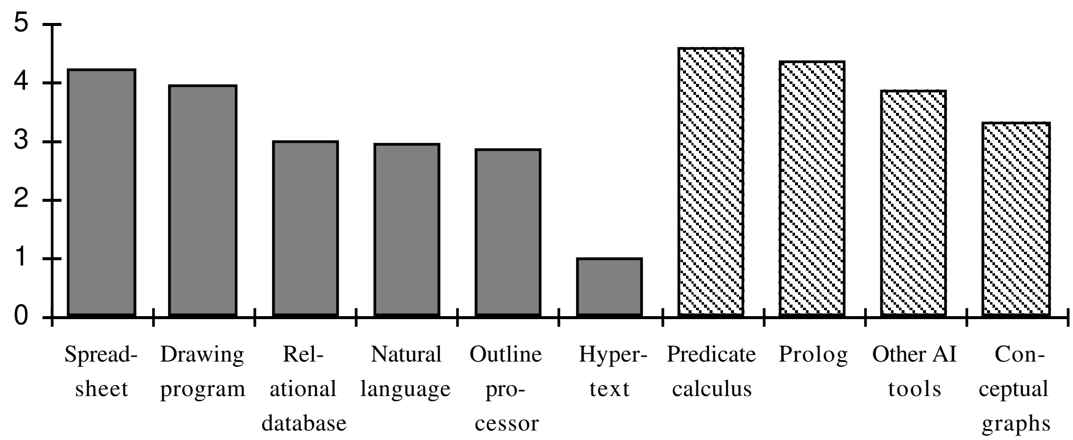


Figure 2: Histogram of perceived difficulty of use (see data in table 2). The technologies on the right (crosshatched) are those from the field of artificial intelligence.

4.3 Conclusions from phase 1: Was success achieved?

We can conclude from the significant numbers of discretionary users of the tool and the positive responses of users in our study, that CODE4 has met the objectives outlined in section 2.2. There would seem, therefore, to be strong evidence in favour of our hypothesis that a simple tool for the manipulation of concepts and properties would prove useful for domain specialists.

As mentioned in section 1.1, however, this does not inform us about exactly what led to success – this is discussed in the next section.

5. Analytic evaluation: Assessing which aspects of the system led to success

Having concluded that CODE4 has met its objectives, we now move on to the second phase of evaluation: Determining what aspects of the tool are the key contributors to that success, and, conversely, what aspects of the tool may be of lesser use, or even detrimental to success.

In this section we present the evaluation of the knowledge representation and user interface features.

5.1 Analysis of the knowledge representation features

We analysed knowledge representation features using the following techniques: a) Studying the knowledge bases to see which representation features are actually used; b) Studying how, in the opinion of users, each feature helps convey the content of each knowledge base, and c) Studying the overall opinions of users about each feature.

Observing which knowledge representation features were present in the resulting knowledge bases

The knowledge bases created by the participants can be analysed by measuring various attributes. Table 3 shows counts of the various classes of concepts in CODE4 knowledge bases.

Class of concept	Total	Mean	Minimum	Maximum	St. Dev.
Main concepts (i.e. types)	2177	91	21	278	61
Instances	182	8	0	55	16
Properties (i.e. relations)	3744	155	52	397	60
Statements (i.e. tuples)	7399	308	34	1614	355
Terms (names for concepts)	6362	265	93	739	168
Metaconcepts (see section 2.3)	209	9	0	80	20

Table 3: Counts of classes of concepts created by participants

The following are some general observations arising from the table 3:

- The number of concepts confirms that substantial work has been done using CODE4. The size of the larger knowledge bases confirms that some of the work was serious in nature.
- The data indicate that users made relatively little use of such features as instance concepts, independent metaconcepts and term concepts. Although their presence may have been useful to a few users, success of the system clearly did not depend on their presence.

In addition to the above counts of concepts, various metrics were developed to better characterize knowledge bases. These are described in Lethbridge (1998). While most of the metrics are out of the scope of this paper, two are particularly relevant:

- On average only 16 percent of the values of properties (called statements in CODE4) were formal, i.e. pointed to other concepts. The rest merely contained text entered by users. Clearly, the ability to provide a mix of formality and informality is important.

- On average only 19 percent of concepts had more than one superconcept. This in fact indicates that multiple inheritance is quite important.

Assessing which features of knowledge representation convey more content

For each knowledge base, users were asked to estimate how much each feature of the knowledge representation contributed to conveying the actual content of the knowledge base (i.e. to expressing its most important information).

The results are shown in tables 4 and 5. Table 4 lists the 14 knowledge representation features about which users were asked. Note that the features are not necessarily disjoint; e.g. users were asked about values (of statements) in general as well as particular types of value.

Group	Aspect	Mean	Max	Min	Std. Dev	Responses
A	The names given to main subjects	9.3	10	7	1.0	21
	The names given to properties	9.1	10	6	1.6	21
	The structure of the inheritance hierarchy	9.1	10	5	1.5	21
B	The structure of the property hierarchy	8.0	10	0	2.5	21
C	Values of statements about main subjects	7.5	10	0	3.2	21
	Informal values as carefully thought-out natural language expressions or sentences	7.2	10	3	2.8	21
	Informal values that are commentary and descriptive	7.2	10	1	3.2	21
D	The layout of graphs	5.9	10	0	3.6	21
	The dimension labels	5.6	10	0	4.0	21
E	Values of statements about metaconcepts	4.7	10	0	3.9	21
	The structure of other relation graphs	4.5	10	0	3.5	21
	Formal values	4.1	10	0	3.7	21
	The order of subproperties in a property hierarchy	3.6	9	0	3.5	21
F	The order of subconcepts in an inheritance hierarchy	2.8	9	0	3.4	21

Table 4: Features of a knowledge bases that convey its content. For each knowledge base, each feature was ranked by the participants on a scale where 0 means unimportant and 10 means essential. The horizontal lines distinguish groups of features that are independently comparable with other groups, as shown in table 5

The features are listed in decreasing order by mean, however there is no statistically significant difference[2] between features whose means are close together. An analysis of where statistically significant differences do exist showed that the features can be divided into eight groups (i.e. equivalence classes), labelled A through H in tables 4 and 5. Table 5 can then be used to determine which groups convey significantly more knowledge than others. For example none of the aspects within group A were considered more important than others, whereas all of group A was considered more important than all of the others (groups B through F). Similarly the single aspect in group B is only more important, in the opinion of participants, than aspects in groups D and F, but not more important than any of the aspects in group C.

[2] According to pairwise t-tests.

	F	E	D	C	B
A	√	√	√	√	√
B	√	√	√		
C	√	√			
D	√				

Table 5: Significant differences among groups of knowledge-conveying aspects. The x and y axes indicate groups of content-conveying aspects from figure 6.6. Wherever a check mark appears, users believed that row aspects were significantly more important than column aspects. Significance was determined using t-tests.

Some general observations about Tables 4 and 5:

- As group A shows, names are considered to be of paramount importance. Users appear to attach great significance to the fact that humans will be using the knowledge base and thus concepts must have effective names. In a purely formal system, operated on by a computer, names are not significant (although it meaningful names are still needed so that humans can work with the formal language).
- Group A supports the intuitive notion that the inheritance hierarchy, with labelled concepts and attached properties, is the fundamental framework for building knowledge bases. Other aspects of knowledge can only be added once a rough inheritance hierarchy is specified.
- The way the property hierarchy is structured has as much importance as the values of statements. This validates CODE4's emphasis of this feature.
- Informal values are considered significantly more important than formal ones (and more important than the graphs of relations that the latter permit to be drawn). This suggests that developing effective ways of dealing with informal values can be a worthwhile exercise. Substantial and useful knowledge bases can be built that contain largely informal knowledge.
- As would be intuitively expected, values of statements about main concepts were considered more important than values of statements about metaconcepts; although metaconcept values were important to some participants.
- Graph layouts and dimension labels, features supported strongly in CODE4, were considered moderately important. Among knowledge bases, however, these features varied in importance.

Assessing general usefulness of knowledge representation features

Users were asked how useful they perceived various knowledge representation features to have been in their work. This set of questions differs from the last set in the following ways:

- The previous set of questions ask about specific knowledge bases, whereas these questions ask for overall impressions of users.
- These questions ask about features in more detail than the previous ones. The result is that fewer users were able to answer any given question.
- These questions ask about usefulness in general, whereas the previous ones focused on how information content was conveyed. These questions use a scale where -5 indicates

that the feature was harmful to their work (i.e. negatively useful), 0 indicates that the feature is of no use, and +5 indicates that the feature is very useful. The previous questions on the other hand, had just required responses on a scale of zero to 10.

Inheritance, in general	**4.9**
Multiple inheritance	**4.9**
The property hierarchy, in general	**4.8**

Table 6: The highest mean responses to the question about usefulness of knowledge representation features.

Data for this set of questions is found in tables 6 (features rated high) and 7 (features rated low). Note that features that were given an intermediate rating are excluded for compactness – their contribution to success or failure of the tool as a whole is considered likely to be neutral. The following are some comments:

- Inheritance and multiple inheritance were judged more important than any other feature, and were classed as 'essential'. This corroborates the conclusion (from the questions about information content) that the inheritance hierarchy is of paramount importance.
- Significantly less important, statistically speaking, was the property hierarchy, however it too was judged very close to essential.
- The next most important feature was multiple parents in the property hierarchy, however no conclusion can be drawn about whether it is in fact more important than the twelve features that follow it in the ranking. Despite this, the emphasis CODE4 places on the property hierarchy appears to correspond with users' needs.
- A large number of features were ranked close together and no conclusions can be drawn about the exact order. In this group are formal and informal values (formal values are ranked marginally higher than informal ones, contrary to findings discussed earlier, but not significantly higher).

Metaconcepts	**3.5**
The ability to add new facets	**3.4**
The ability to have more than one term for a concept (synonyms)	**3.2**
The ability to use almost any character string in a term (the name of a property or main concept)	**3.2**
Treating terms, properties and statements as full-fledged concepts (with their own properties etc.)	**3.1**
Automatic combination of conflicting values in multiple inheritance	**2.8**
Delegation	**2.7**
The set of built-in facets	**2.6**
Instance concepts	**1.9**
Terms as separate concepts from the concept(s) they designate	**1.9**
The maintenance of information about which concepts are disjoint	**1.5**
Treating facets as properties	**1.4**
The fact that you need not explicitly name a concept (automatic default naming)	**0.2**

Table 7: The lowest mean responses to the question about usefulness of knowledge representation features.

- Ranked close together at a moderate level of importance were several features associated with naming (synonyms, unrestricted syntax for names, and treating terms as concepts). However the feature of CODE4 that results in automatic default naming was ranked far lower (close to ‘unimportant’). The latter might be because users almost always name a concept as soon as they create it, so they consider the default name of little use. However, generating a default name permits the interface to be non-modal (this was ranked highly in a question discussed in the next subsection).
- Although no features were given an overall assessment of ‘harmful’ (i.e. negatively useful), three features were given such an assessment by individual users. These were automatic combination under multiple inheritance, treating facets as properties and automatic default naming.

5.2 Analysis of the user interface features

This section continues the analytical evaluation of CODE4 by considering user interface features. Two types of questions were posed in order to perform the evaluation: The first type of question asked subjects to give their general opinion about the usefulness of features; the second type of question probed how much they used certain mediating representations, where alternatives were available.

Assessing general usefulness of knowledge representation features

In a similar manner to the questions about knowledge representation features, users were asked a series of questions about user interface features. They used the same scale as previously (where -5 means harmful and 5 means essential).

The outline mediating representation, in general	**4.8**
Dynamic updating of subwindows	**4.7**
Direct typing to change a concept name in the outline and graphical mediating representations	**4.6**
The graphical mediating representation, in general	**4.6**
The ability to save various graph layouts	**4.5**
The control panel, in general	**4.5**
The ability to make multiple selections on the graphical and outline mediating representations	**4.5**
Relation subwindows (e.g. showing part-of or other arbitrary relations of combinations of relations)	**4.3**
The matrix mediating representation, in general	**4.3**
Different mediating representations (outline, graphical, matrix) with commands that work in a similar manner in each	**4.3**
The variety of ways of making or extending a selection (dragging, marquee, shift key, control key)	**4.2**

Table 8: The highest mean responses to the question about usefulness of user interface features.

Most of the features about which questions were asked are either novel in CODE4 or are emphasized far more in CODE4 than in other knowledge management technology. Data is found in tables 8 and 9; the following are some specific observations:

- Four features were ranked as close to essential and were significantly more important than most of the others. Foremost of these was the outline mediating representation.

Close behind this was: a) dynamic updating of windows, b) direct typing to change a name and c) the graphical mediating representation.

- Ranking significantly behind the above four, but still 'very important' were the ability to save multiple graph layouts, the control panel and the ability to make multiple selections. These features are not 'new ideas', but their inclusion appears to greatly enhance the system.
- A very large number of features were ranked indistinguishably close to each other as 'important'. Examples are the non-modality of the interface, the matrix mediating representation, relation knowledge maps and features concerned with the mask.
- No features were ranked as 'unimportant' or 'harmful' but a few features were ranked as only slightly useful (table 9). These included the ability to attach a graphical icon to a concept and the ability to have multiple knowledge bases loaded at once. The latter feature was ranked as harmful by one user.

The options available in the control panel (e.g. the format options)	**3.4**
Unlimited chains of driving and driven subwindows	**3.4**
The ability to arrange an outline both alphabetically and hierarchically	**2.6**
The ability to have multiple knowledge bases loaded at once	**2.6**
The ability to show placeholders for hidden concepts (+/-)	**2.1**
The ability to attach a graphical icon to a node in the graphical mediating representation	**1.2**

Table 9: The lowest mean responses to the question about usefulness of user interface features.

Assessing the extent to which users used particular mediating representations when entering knowledge

For each knowledge base, users were asked to estimate how much of the knowledge was entered using the four major mediating representations. In agreement with the above general questions, users reported using the outline representation far more than any other.

The matrix representation was used relatively little, although it was developed after some users had started their work. Also, it is not (yet) possible to *start* creating a knowledge base using a matrix window; it can only be used as a query mechanism or to fill in missing values.

Table 10 gives the data about mediating representation usage. The statistics for the 'user language' mediating representation and the outline representation are likely to be too low and high respectively. This is because many users are not aware that the user language representation is considered distinct from the outline representation (the former, which is used only for entering a single property value, is usually associated with an outline but can be associated with a graph).

Mediating Representation	Mean	Max	Min	Std. Dev
Outline	64.4	100	30	22.8
Graphical	14.8	50	0	18.7
User language	16.3	50	0	16.7
Matrix	4.6	20	0	6.1

Table 10: Amounts of knowledge entered using different mediating representations.

5.3 Tasks performed during knowledge management

The previous two sections have separately analysed the various features of CODE4 to determine which are apparently most useful to the system's success. This section continues the analysis by looking at various aspects of the work they performed using the tool – tasks that involve both knowledge representation and user interface.

How difficult do users find various knowledge management tasks?

For each knowledge base, users were asked how difficult various tasks were to perform. Difficulty was ranked on a scale where zero indicates extremely easy and ten indicates extremely difficult.

The data is shown in tables 11 and 12. Table 11 orders tasks by mean. Table 12 shows which groups of tasks were significantly more difficult than others. The following are general observations:

- None of the tasks was judged to be extremely difficult; most of the tasks were closer to 'easy' than to 'difficult'. This suggests that CODE4 is helping users.
- The only possible exception was 'Determining whether a part of the knowledge base is correct, coherent and consistent'. This task was judged significantly more difficult than almost all others. This suggests the need for more facilities to provide feedback.
- Judged to be less difficult than the above, but still moderately difficult, were two 'understanding' tasks: understanding the domain, and understanding CODE4's error messages. The latter again suggests that better feedback facilities are needed.
- Several tasks related to specifying properties are rated moderate (neither easy nor very difficult). These tasks include naming a property, dealing with several properties that have the same name, determining the most general subject for a property, determining whether a property is already present or not, and moving a property to a new most general subject. This suggests that a 'property assistant' tool would prove useful.
- Tasks that are largely mechanical in nature were judged reasonably easy by the participants. Such tasks included reparenting (i.e. changing a superconcept or superproperty) and setting up windows.
- Two tasks were judged significantly easier than the others: These were finding a main subject that already exists and understanding the effects of inheritance. This helps show that CODE4's querying and display capabilities are effective.

Group	Knowledge management task	Mean	Max	Min	Std. Dev	Responses
A	Determining whether a part of the knowledge base is correct, coherent and consistent	5.8	10	2	2.4	23
B	Understanding the subject matter being entered into the knowledge base	5.2	10	0	3.2	23
	Understanding what is the problem when the system does not permit a command	5.1	10	0	3.2	23
C	Naming a property	4.3	8	0	2.9	23
	Determining the interrelationships among a particular set of concepts	4.2	9	1	2.5	19
D	Determining whether a part of the knowledge base is complete enough	4.2	8	1	2.9	20
	Dealing with the situation where there are several properties with the same name which should really have been the same property	4.2	9	0	2.9	20
E	Figuring out where to put a concept in the inheritance hierarchy	4.1	8	1	2.2	23
	Figuring out the most general subject for a property	4	9	2	1.8	23
F	Specifying a special relationship (e.g. part-of) between concepts	3.8	9	0	2.9	22
G	Determining whether a property is important enough to add	3.6	8	1	2.0	23
	Determining whether a property exists already or not	3.3	8	1	2.1	22
	Changing a character string (possibly in many places)	3.3	10	0	3.3	23
H	Determining whether a concept is important enough to add	3.2	8	1	1.8	23
H	Filling in a value, in general	3.1	10	0	2.1	23
I	Moving a property to a new most general subject	3.1	9	0	3.5	22
H	Figuring out where to put a property in the property hierarchy	3.1	7	1	1.9	23
I	Reparenting a property	3.1	9	0	3.5	20
I	Reparenting a concept	3.0	9	0	3.2	21
H	Determining on what statement to put a value	3.0	7	0	2.0	21
H	Setting up windows to show the knowledge in which you are interested	2.8	10	0	2.8	23
I	Naming a main subject	2.7	7	0	2.6	23
I	Finding a property that already exists in the knowledge base	2.6	7	1	2.1	21
J	Finding a main subject that already exists in the knowledge base	1.9	7	0	1.7	21
	Understanding how inherited values get their content	1.8	9	0	2.3	23

Table 11: Users' perceptions of the difficulty of tasks. For each knowledge base, each task was ranked by the participants on a scale where 0 means extremely easy and 10 means extremely difficult. The horizontal lines distinguish groups of tasks that are independently comparable with other groups, as shown in table 12. Groups H and I could not be separated by a unique line.

	J	I	H	G	F	E	D	C
A	√	√	√	√	√	√	√	√
B	√	√	√	√				
C	√	√	√					
D	√	√						
E	√	√	√					
F	√	√						
G	√							
H	√							
I	√							

Table 12: Significant differences among groups of knowledge management tasks. The x and y axes indicate groups of knowledge management tasks from figure 6.6. Wherever a check mark appears, users believed that row tasks were significantly more difficult than column tasks. Significance was determined using t-tests.

Assessing features that support formality vs. features that support informality or spontaneity

One of the major tenets in the design of CODE4 was that users should be able to represent knowledge both formally and informally. Studies of knowledge representation and user interface features reported earlier show that users feel the need for both formality and informality. The following questions were designed to obtain further evidence about this:

Participants were asked to judge the degree of formality of their knowledge bases on a scale where 0 means completely informal and 10 means completely formal. The mean response was 4.8 with a standard deviation of 2.7. Responses ranged from 0 to 9 for the 24 knowledge bases from which responses were received. Interestingly, individual participants gave very different responses for different knowledge bases they created. It thus appears that informality and formality are both necessary, although as was discussed earlier, users feel the informal aspects might convey more knowledge.

Participants were also asked to indicate how rapidly they would represent an idea when it occurred to them. The principle here is that informal capabilities should allow users to rapidly record their thoughts, if they so wish. The scale used was as follows: 0 means that they would represent the idea immediately and later on worry about whether it was correct; and 10 means that they would never represent the idea until they were absolutely sure they could do it correctly. The mean response was 4.2, with a range of 0 to 8 and a standard deviation of 2.4. Being in the middle of the range, this suggests that users need both facilities to rapidly enter ideas, and also facilities to rapidly help them decide whether they *should* enter an idea.

How difficult were naming tasks?

Users were asked what percentage of main subjects or properties did not have standardized terms; i.e. they had trouble putting a term on a concept, deciding if two terms had the same meaning or even *what* a term meant. The mean response was 29% (range 10% to 90%;

standard deviation 22%). This indicates naming is not a straightforward task, and justifies CODE4's attempts to provide better facilities to handle names.

As table 11 shows, users found naming properties to be significantly more difficult than naming main subjects.

Users were also asked what percentage of main subjects and properties had a name that the participant invented, as opposed to one that might be found in an ordinary dictionary or technical glossary. The mean was 17% (range 0% to 70%; standard deviation 18%).

5.4 Conclusions from phase 2: Which aspects of the system led to success?

The following summarizes the results of evaluating the importance of features – the process we call analytic evaluation. It is important to note that many features of CODE4 are interdependent. For example, users did not directly judge the uniformity of concepts to be important; they nevertheless found facilities to deal with terms, metaconcepts and properties to be useful. Treating concepts uniformly allows the latter to be designed more easily; such treatment also facilitates browsing and other useful capabilities.

- **Features for dealing with the inheritance hierarchy**: Users judged the inheritance hierarchy to be one of the most generally important features and to convey much of the knowledge in their knowledge bases. While this is not unexpected, it suggests that knowledge management systems must provide effective facilities, like CODE4's outline mediating representation, for rapid manipulation of this hierarchy.
- **The property hierarchy and features for dealing with it**: Users found these both generally important and useful in conveying knowledge.
- **Features for dealing with terms and naming**: Users found names of properties and main subjects to be very important in conveying meaning. They also found naming to be intrinsically difficult since they had to frequently invent names or choose among several alternatives.
- **Features for handling informality**: Users judged that informality was important in conveying much of the knowledge – although not to the exclusion of formality.
- **The non-modality of the interface and dynamic updating**: These features were rated important by users.
- **Mediating representations**: Users rated all three major mediating representations as useful, particularly outlines. They considered the ability to save graph formats to be important.
- **Search facilities**: Users indicated that finding concepts, a task frequently performed using the mask, was one of the easiest tasks to perform.

On the other hand, there were a variety of features that we judge did not contribute to the success of the tool. These include instance concepts, combination, delegation and a variety of formatting options. These nevertheless involved considerable development work.

The following needs were identified during the evaluation.

- **Tools to help specify properties**: Users found naming properties to be particularly difficult. They also found manipulating them to be difficult and in need of support.

- **Tools to assist with analysis**: Since users reported that one of their most difficult problems was evaluating correctness and coherence of the knowledge base, there is an obvious need for analysis tools. The metrics, which were unavailable to users when they created their knowledge bases, might help in this process.
- **Improved facilities for feedback**: Users found that understanding CODE4's error messages was one of the most difficult tasks.

6 Summary and Conclusions

This paper has presented the evaluation of a knowledge management tool called CODE4, as well as techniques for performing and presenting such evaluations in general. The design of the tool was presented in terms of task analysis, objectives and design rationale. The evaluation was then split into two sections: Effectiveness (judging success) and analytic (determining what led to success or failure).

The tool was judged to be effective because a) users from the target population (domain specialists) regarded developing their knowledge bases to be a worthwhile exercise; b) the users used CODE4 repeatedly and produced significant knowledge bases with it, and c) the users judged CODE4 to be easier to use than other tools for knowledge management.

With regard to the reasons for success, it appears the most important contributors were features that allowed easy direct-manipulation interaction with hierarchies, the names of items and a combination of formal and informal knowledge.

References

Bowker, L and Lethbridge, T.C. (1994a). "Terminology and Faceted Classification: Applications Using CODE4", *Advances in Knowledge Organization* (Proc. Third ISKO Conference), Copenhagen, June, pp. 200-207.

Bowker, L and Lethbridge, T.C. (1994b). "CODE4: Applications for Managing Classification Schemes", *proc . 5th ASIS SIG/CR Classification Research Workshop*, Alexandria, Virginia, October

Bradshaw, J., J. Boose, D. Shema, D. Skuce and T. Lethbridge (1992). "Steps Toward Sharable Ontologies for Design Rationale". *AAAI-92 Design Rationale Capture and Use Workshop*, San Jose, CA, July, pp. 29-38.

Bradshaw, J., Holm, P., Boose, J., Skuce, D., and Lethbridge, T.C. (1992, October). "Sharable Ontologies as a Basis for Communicating and Collaborating in Conceptual Modeling". *Proc. 7th Knowledge Acquisition for Knowledge-Based Systems Workshop.* Banff, Alberta, pp. 3.1-3.25

Diaper, D. ed. (1989). *Task Analysis for Human-Computer Interaction.* Chichester: Ellis Horwood.

Eck, K. (1993) *Bringing Aristotle Into the Twentieth Century: Definition-Oriented Concept Analysis in a Terminological Knowledge Base*, Master's thesis, University of Ottawa.

Ghali, N. (1993) *Managing Software Development Knowledge: A Conceptually Oriented Software Engineering Environment*, M.Sc. Thesis, University of Ottawa, Dept. of Computer Science.

Hix, D., and Hartson, H.R. (1993). *Developing User Interfaces: Ensuring Usability through Product and Process*. New York: John Wiley.

Lethbridge, T.C., and Skuce, D. (1992), “Informality in Knowledge Exchange”. *Proc. AAAI-92 Workshop on Knowledge Representation Aspects of Knowledge Acquisition.* San Jose, July, pp. 93-99.

Lethbridge, T.C. and Skuce, D. (1994). “Knowledge Base Metrics and Informality: User Studies with CODE4”. *Proc. 8th Knowledge Acquisition for Knowledge-Based Systems Workshop.* Banff, Alberta, January, pp. 10.1 - 10.19.

Lethbridge, T.C. (1994). *Practical Techniques for Organizing and Measuring* Knowledge, Ph.D. Thesis, University of Ottawa

Lethbridge, T.C. (1998, to appear). “Metrics for Concept-Oriented Knowledge Bases”, *International Journal of Software Engineering and Knowledge Engineering*, **8** 2, June.

Meyer, I., K. Eck and D. Skuce (1994). "Systematic Concept Analysis Within a Knowledge-Based Approach to Terminology", in *Handbook of Terminology Management,* S. E. Wright and G. Budin eds. Amsterdam/Philadelphia: John Benjamin.

Skuce, D. (1993). "A System for Managing Knowledge and Terminology for Technical Documentation". *Third International Congress on Terminology and Knowledge Engineering*, Cologne, pp. 428-441.

Skuce, D. and Lethbridge, T.C. (1995). “CODE4: A Unified System for Managing Conceptual Knowledge”. *International Journal of Human-Computer Studies* 42, 413-451.

Wang, C. (1994) *Towards Conceptually-Oriented Software Requirements Analysis and Design*, M.Sc. Thesis, University of Ottawa.

Whiteside, J., Bennett, J., and Holtsblatt, K. (1988). Usability Engineering: Our Experience and Evolution. In *Handbook of Human-Computer Interaction* (Helander M. ed.). Amsterdam: North-Holland.