KNOWLEDGE BASE METRICS AND INFORMALITY: USER STUDIES WITH CODE4

Timothy C. Lethbridge
Doug Skuce

Department of Computer Science
University of Ottawa, Canada K1N 6N5
{tcl, doug}@csi.uottawa.ca

ABSTRACT

We describe studies of the CODE4 knowledge management system. Two thousand hours of CODE4 use by 8 users are analysed to validate various design hypotheses. We discuss initial ideas for some knowledge base metrics and we show that users place a very high value on certain features of the knowledge representation and user interface, such as those that permit informal use.

1. INTRODUCTION

Many researchers have designed knowledge representations and built systems for the construction and maintenance of knowledge bases. For the most part the evaluation of such work has been by one or more of the following methods:

- Appealing to mathematics: Researchers have shown how their representations have interesting properties such as a logic-based formal semantics and the ability to support specific types of logical inference
- Appealing to theories from psychology and related disciplines.
- Demonstrating solutions to interesting but specialized general problems.
- Showing how a small number of problematic special cases can be handled.

An approach that has not been tried frequently is to analyse larger numbers of scenarios where users work on complete projects from start to end. In this paper we take this approach in order help establish the usefulness of our work and how well it 'scales up'.

We use the above approach on CODE4, a tool we have developed for the acquisition and manipulation of knowledge. CODE4 features a high degree of flexibility in its knowledge representation and user interface, and has been designed so that 'domain experts' can easily and rapidly construct knowledge bases after only a few days training. We describe CODE4 in detail in a companion paper in this proceedings (Skuce and Lethbridge, 1994).

The overriding hypothesis which we have sought to validate by developing CODE4 is that it is possible for ordinary people to use knowledge acquisition technology for a wide variety of tasks[1]. The fact that CODE4 and its predecessors have attracted on-going interest

---

[1] We do not claim that we can magically solve many of the very difficult and real problems in knowledge acquisition that are focussed on most heavily by other researchers, such as how to extract years of implicit knowledge from an expert. What we do propose though, is that there are many

from users in a number of occupations (linguists, organizational modellers, philosophers, software engineers, documenters etc.) goes a long way towards validating this top-level hypothesis.

In order to improve CODE4, we have formulated several hypotheses about how the tool can be made more accessible and useful; we have then sought to validate these hypotheses by user testing. In the next section we present our research method. In section 3 we describe some metrics that help us describe the nature of our knowledge bases. In the final section we describe users' responses to key features in CODE4; many of these features allow for informal use of the system.

2. RESEARCH METHOD

CODE4 is the successor to prior versions of CODE, most notably CODE2 (Skuce et al, 1989). CODE2 has been used extensively in several projects in industry and academia. This experience taught us some useful lessons and led to the development of a number of hypotheses. Some of the key general hypotheses are:

- A non-modal, graphical and flexible user interface is critical to the effectiveness of such a system.
- A flexible, elegant knowledge representation is important.
- Features that permit informal use are at least as important as features intended for more structured, systematic representation work.

Most of the hypotheses were developed by observing problems with CODE2 and CODE3 and analysing the underlying reasons for the problems. In January 1991 it was decided to start construction of CODE4, which incorporates many of the hypotheses as design principles.

Since mid-1991 CODE4 has been used by a growing number of people and during this time new features have been prototyped and incorporated. Important projects using CODE4 are a group doing organizational modelling at Boeing (Bradshaw et al. 1992) and a group of translation researchers investigating tools to handle technical terminology (Meyer et al, 1992). In mid-1993 the system reached a level of maturity sufficient to begin more structured studies of its effectiveness.

We gave CODE4 to eight subjects (graduate students and a professor). The subjects created a total of 18 knowledge bases using the system. They reported spending a total of 2000 hours building the knowledge bases, in total representing over 16500 concepts[2].

We use four methods to analyse the subjects' efforts; two are objective in nature and two subjective:

---

simpler tasks to which one can apply KA technology. The challenge is to make that technology accessible. Very few KA researchers are dealing with this particular problem.

[2] CODE4 handles all units of knowledge uniformly, and refers to them all as *concepts*. The figure 16500 represents the number of concepts users *added* to knowledge bases. A number of primitive concepts exist when a knowledge base is created.

1) Static analysis of the knowledge bases: We measured over 100 attributes of each knowledge base; a few of these are reported in table 1. These attributes are the basis for the metrics discussed in section 3.

2) Analysis of the user interface commands used to create each knowledge base: Throughout the period of the experiment, all usage of CODE4 at the University of Ottawa was logged in detail – these data were then analysed to determine which commands and operations appear most useful.

3) Analysis of a detailed questionnaire administered to each subject. The questionnaire contains 55 main questions and 389 sub-questions. Most of the questions are designed to ascertain the usefulness (or importance) and difficulty of various features. Figures 1 and 2 show samples of questions.

4) Follow-up interviews with the subjects. Here we tried to further our understanding of people's problems, and why they did certain things. In several cases, misconceptions about CODE4 or the questionnaire were highlighted by this process.

The results reported in this paper come largely from methods 1 and 3: the static analysis and the questionnaire. The questionnaire gives us immediate feedback about how successful we have been at achieving our goal; results from its analysis are found largely in section 4 where we discuss features of CODE4. The metrics developed from the static analysis (section 3) gives us fewer immediate clues as to how well we have achieved our goal, but they provide a baseline for continual improvement, and provide a a way for others to evaluate their systems.

When analysing the data resulting from the questionnaire and from the static and command analyses, we used statistical tests such as the t-test and tests for correlation. We used 95% confidence intervals to test for significance. For a substantial number of the questions we were not able to obtain results within such an interval, so those questions have been omitted from discussion in this paper.

Due to the diversity of backgrounds of the users, the complexity of CODE4 and the interaction among its features, there is limited scope to run controlled experiments of the type where specific features are isolated. We therefore seek to validate our hypotheses using collaborating data from related questions or from more than one of the analysis methods.

Class of concept	**Total**	**Mean**	**Minimum**	**Maximum**	**St. Dev.**
Types (includes 4 primitives each)	2007	112	40	288	60
User instances	116	6	0	51	14
User props (inc. 34 primitives each)	2502	172	55	400	60
Statements	6398	355	34	1614	391
Terms	5444	302	94	739	177
Metaconcepts	209	12	0	80	22
All concepts: M_{TOT}	17266	959	224	2825	653
Main subjects: M_{MSUBJ}	1920	107	30	278	61
Statements per main subject		*2.10*	*0.67*	*4.15*	*1.01*
Terms per main subject		*1.04*	*1.00*	*1.20*	*0.05*

Table 1: A few of the basic attributes of the 18 knowledge bases created by the 8 subjects. The reader should consult (Skuce and Lethbridge, 1994) for more details about these classes of concept.

' □ In this question, please rank your perceptions of various knowledge representation features. Please use the same scales as in question C1.

	i. Familiarity? 0 .. 10	ii. Useful to you? -5 .. 5	iii. Useful in general? -5 .. 5
a) Metaconcepts.....			
b) Instance concepts.....			
c) Terms as separate concepts from the concept(s) they designate.....			
d) Treating terms, properties and statements as full-fledged concepts (with their own properties etc.).....			

Figure 1: Sample questions asked of the subjects about CODE4's knowledge representation in general.

(□□ In the texts or other sources you consulted, what percentage of the concepts or properties did not have standardized terms (e.g. you had trouble putting a term on a concept, deciding if two terms had the same meaning, or even *what* a term meant?)

KB1: _____ KB2: _____ KB3: _____

(□□ What percentage of main concepts and properties have a name *you* invented as opposed to one that might be found in an ordinary dictionary or a technical glossary?

KB1: _____ KB2: _____ KB3: _____

Figure 2: Sample questions from the questionnaire, asking about individual knowledge representation projects.

3. THE NATURE OF KNOWLEDGE BASES: SOME METRICS

In this section we discuss objective measurement of knowledge bases. We have several purposes in developing metrics:

1) To understand how knowledge bases, users and domains differ. We hope to detect patterns in the data, and correlations between the metrics and the answers from questionnaires.
2) To understand the structure of individual knowledge bases so we can develop user interface and knowledge representation features that better cope with such structures.
3) To give us baselines for continual improvement.
4) To provide means whereby research using different systems can be put on a common footing.

The metrics described in this paper are only preliminary suggestions; we have found them useful in our research but are not yet ready to propose them as standards.

We approach our analysis of metrics by first defining the kind of knowledge we are measuring and then describing the kinds of general measuring tasks we may want to perform. After this we discuss how to evaluate the metrics, asking the question: what features are desirable or undesirable in a metric? In the final part of the discussion, section 3.4, we

present the metrics themselves: We discuss their advantages and disadvantages and outline a few things they tell us about CODE4 and its users.

3.1 What kind of Knowledge Bases are we Measuring?

For the purposes of this study we are concerned with frame-based or semantic-net based representations such as CODE4-KR, KM and KL-ONE. These are now often called object oriented representations. We believe however that the basic principles can be adapted to rule-based representations or representations of problem-solving methods.

The reason for the latter assertion is that object oriented representations can be made to subsume the others; or the latter can be seen as more specialized abstractions for the purposes of certain tasks. For example, when representing rule-oriented knowledge in CODE4-KR one might encode the rules as instance concepts and have additional concepts for 1) the things affected by these rules, 2) properties of the rules, 3) statements about them etc.. One can likewise imagine encoding problem solving methods in an object-oriented way.

3.2 Measuring Tasks to be Performed on Knowledge Bases

The following subsections list tasks that involve measuring attributes of knowledge bases. many of the tasks have analogs in conventional software engineering, but others do not. In this section we explain the motivation for the measuring tasks. Suggestions for actual metrics and examples of their use can be found starting in section 3.4.

The measuring tasks can be divided into three rough categories: Predicting tasks, static judgement tasks (for completeness, complexity, information content and balance), and comparison tasks (between knowledge bases, domains, knowledge acquisition techniques and knowledge representations). The tasks are certainly not independent – most tasks depend explicitly on others – however, categorizing measuring tasks gives us a useful basis to decide what metrics we should develop.

We should not expect a clear mapping between tasks and metrics: Some metrics might help with several tasks, and tasks may require several metrics.

Task A – Predicting time to completion, and hence cost: This is one of the main tasks for which software engineering metrics are developed: People are interested in ascertaining the amount of work involved in a development project so they can create budgets and schedules, or so they can decide whether or not to go ahead with the project as proposed.

The following illustrates the general scenario for such metrics; here we use the term ‘product’ to stand for either ‘software’ or ‘knowledge base’:

a) We have measured certain attributes of a number of products $P_1..P_{n-1}$.

b) We are interested in making predictions about P_n.

c) We assume certain metrics $M_1..M_m$ are relatively constant among $P_1..P_n$

d) Another metric, M_s, is fairly simple to measure early in development.

e) Another metric, M_t, represents time to create the knowledge base.

f) By analysing $P_1..P_{n-1}$, a function, f_p, is developed that reasonably accurately predicts M_t, given the constancy of $M_1..M_m$, i.e. $f_p(M_s) \rightarrow M_t$.

In the software engineering world, a popular early candidate for M_s is function points (Low and Jeffrey, 1990). Another, somewhat weaker M_s is lines of code. Several methods including COCOMO (Boehm, 1981) fulfil the role of f_p, predicting the number of person-months to complete the project, given M_s.

To our knowledge, no similar metrics have been published for the production of knowledge bases. Many people consider the production of knowledge bases to be a case of software engineering (especially when a KB is being developed for use in an inference oriented system such as an expert system), however it is intuitively apparent that lines of code or function points have little meaning and less predictive value for knowledge bases. Furthermore few other attributes of standard software engineering projects have similar values in knowledge engineering, rendering attempts to use COCOMO or similar techniques rather useless.

For knowledge engineering, we need to come up with new candidates for M_s and new functions for f_p. Furthermore, we need to ascertain those attributes $M_1..M_m$ that need to be constant for our particular M_s to work effectively (Of course if an attribute is not constant in general, then we can attempt to measure it and incorporate it into our calculation of M_s).

Task B – Judging completeness: In standard software engineering a product is typically considered complete if it fulfils its specification (e.g. passing appropriate testcases). There are a number of fundamental differences however, between standard software engineering and knowledge engineering, that make it necessary to find different ways of determining completeness:

- There rarely is a specification; knowledge engineering typically progresses fluidly towards a goal, perhaps using a prototyping approach.
- Whereas standard software can be broken down into well-defined modules with well-defined outputs that can be tested; most knowledge bases have a much finer grain – being composed of concepts or rules.
- Knowledge can be expressed in a rough form that is *partially* suited to the task at hand, and then gradually refined and formalized so that it becomes more suited to the intended task. Standard software tends to either work or not work so there is less scope for incremental development at the detailed level.

The standard way of measuring KB completeness is to apply a knowledge-based system to a set of test tasks that have optimum expected outcomes, and to measure how well the system performs. This has a number of disadvantages: a) It is typically hard to measure the completeness of sub-components of the knowledge base, since for correct operation most components of knowledge based systems must operate together. b) In incremental development it is all too easy to refine just those areas of the KB that lead to good performance on the test set, and thus to bias the system. c) For some types of knowledge bases it is not feasible to come up with meaningful test tasks; for example if the knowledge base represents the *design* of something, or is intended to serve as an educational resource that students explore in order to learn about some domain.

The above points imply that there may be scope for developing completeness metrics that do not rely on operational testing: For example, we may be able to measure a knowledge base's degree of refinement or formalization. Such metrics would not predict whether the knowledge is in any sense *correct*, but they might point out when and where further work would be profitable, and perhaps how much.

Task C – Judging complexity: A number of metrics allow software engineers to determine the complexity of modules or subsystems. There are several reasons for doing this: High complexity may be predictive of greater cost in future stages of development; and it may expose situations where designs can be improved. A measure of complexity may also help in determining productivity, because a small but complex project may take as much work as a large but simple project.

In software engineering one might measure cyclomatic complexity, coupling, fan-out and hierarchy depth. Adaptations of the latter three may well apply to knowledge bases, but the special nature of KB's may suggest additional useful metrics.

Task D – Judging information content: It is not usual to ask of conventional software a question like: How much information is in this system? In general, such a system is designed to be able to perform a limited number of tasks. With many knowledge bases however, a major goal is that they be queriable in novel ways – deducing new facts using various inference methods. We want to be able to uncover latent knowledge.

To be able to estimate information content helps us determine both the potential usefulness of a knowledge base and how productive its development effort is. Such metrics should take into account the fact that some knowledge is a lot less useful than other knowledge.

Task E – Judging balance: Knowledge bases are typically composed of a mixture of very different classes of knowledge; e.g. concepts in a type hierarchy, detailed slots, meta-knowledge, commentary knowledge, procedural knowledge, rules, constraints etc. Each project may require a different *balance* of these classes; and various metrics might give us clues as to what kind of balance we have achieved. Other knowledge base measures that come under the category of balance are those that show whether different parts of a knowledge base are equally complete or complex, i.e. whether completeness and complexity are focussed in certain areas.

There is likely to be a strong relationship between measures of completeness and measures of balance; and some balance measures may help create a composite measure of completeness. For example if a knowledge base has a low ratio of rules to types, it may imply that there is scope to add more rules; likewise if one subhierarchy is far more developed than another the overall KB may be incomplete. However, there are reasons for measuring balance variables separately (perhaps after a subjective judgement of completion): They can allow us to characterize the nature of knowledge and to classify knowledge bases. In the metrics described in this paper, we omit balance to conserve space but believe measures of this type to be useful.

There are not many analogs to the idea of balance in general software engineering, but one example is measuring the ratio of comment lines to statement lines.

Task F – Comparing knowledge bases: Most of the metrics derived from tasks listed above can lead to useful differential measures between particular knowledge bases. For example we can determine how much more complex KB A is than KB B.

Task G – Comparing domains: By measuring various attributes about knowledge bases in particular domains, we may be able to ascertain certain constant factors that characterize each domain, distinguishing it from others.

This kind of knowledge can feed back into the prediction process (task A). For example in the COCOMO method, different predictive formulas are applied to embedded software and to data processing software. Similarly we may be able to distinguish features of classes of knowledge base that would allow us to create different prediction formulas (or different coefficients in the same formulas) for, say, medical rule based systems, electronics diagnosis systems and educational systems.

Task H – Comparing development techniques: By comparing knowledge bases developed using different knowledge acquisition techniques we can decide which techniques are better for certain tasks. We can also incrementally improve techniques.

Task I – Comparing representations: In a similar manner to comparing development techniques, we can compare representations. Even though such a process might require converting each representation to some common denominator, we might be able to gain useful knowledge about the effectiveness of the abstractions in each representation.

3.3 Criteria for Evaluating Metrics

The following are some questions one should ask when evaluating metrics. These criteria are used in the next section when we discuss actual metrics we have developed:

- How subjectively useful is the metric? The metric must perform some useful task.
- How intuitive is the metric? To be preferred are simpler metrics and ones where people easily understand the reasons for the calculations.
- How correctly does the metric measure what we want? We must be sure that the calculated number corresponds to what we are trying subjectively to measure. If the objective phenomena being measured do not correspond to the subjective phenomena it might be because some objective factor is omitted or incorrectly weighted.
- How independent is the metric from others? Sometimes this can only be determined by actually measuring knowledge bases and testing for correlation. Often however, it is possible to ascertain mathematically that certain dependencies exist. Ideally metrics should be as independent as possible.
- What are the logical extreme cases of the metric? Do these correspond with what one would intuitively expect, or are there pathological cases? If the metric behaves in an undesired manner at the ends of its ranges, then we probably should modify it. A metric that has fixed extremes might have more overall usefulness than an open-ended metric; and there should be some intuitive meaning for the ends of the metric's range. In the metrics discussed below we have attempted to find formulas that give a zero-to-one range. This allows easy comparison between metrics
- How sensitive is the metric to changes in the subjective impression it is trying to measure, and to what degree of precision should we measure it? A metric should make

appropriate distinctions without excessive noise. Also a metric should have a similar sensitivity throughout its range.

- How should the metric be weighted when creating composite metrics? For example when creating a measure of overall complexity we have to combine several factors, and decide on the relative importance of those factors. For the composite metrics discussed below we have weighted all factors equally (due to lack of enough data to justify any other weighting).

3.4 Metrics for CODE4 and Similar Knowledge Bases

In this section we discuss several metrics we have developed. Statistics for these are summarized in table 2. There are three classes of metrics: 1) general open-ended measures of size, 2) measures of various independent aspects of complexity, and 3) compound metrics.

Metric	Potential Range	Mean	Min	Max	Std. Dev.
Raw size measures					
All concepts: M_{TOT}	(40-∞)	959	224	2825	653
Main subjects: M_{MSUBJ}	(0-∞)	107	30	278	61
Independent complexity measures					
Userprop Ratio: M_{UPMS}	(0-1)	0.30	0.05	0.70	0.18
Detail: M_{DET}	(0-1)	0.14	0.03	0.46	0.10
Statement Formality: M_{SFORM}	(0-1)	0.16	0.00	0.48	0.17
Diversity: M_{DIV}	(0-1)	0.73	0.09	0.99	0.26
Second Order Knowledge: M_{SOK}	(0-1)	0.08	0.00	0.41	0.12
Isa Complexity: M_{ISA}	(0-1)	0.43	0.29	0.59	0.09
Multiple inheritance: M_{MI}	(0-1)	0.13	0.00	0.87	0.21
Compound complexity measures					
Apparent completeness M_{ACPLT}	(0-100%)	35%	10%	63%	15%
Pure complexity: M_{PCPLX}	(0-37)	1.00	0.35	2.22	0.47
Overall complexity: M_{OCPLX}	(0-∞)	114	18	360	99

Table 2: Statistics for the metrics discussed in this section, gathered from knowledge bases created by the users in our study.

Metrics for size: One of the most basic questions we need to ask is: How ‘big’ is this knowledge base; what is its ‘size’? Knowing this can help us predict development time and judge information content. But what does ‘big’ or ‘size’ mean? In software engineering, size can be measured in lines of code (LOC), unadjusted or adjusted function points, projected time required etc. The former measure is often criticised as being too dependent on implementation and design, but still gives a very concrete baseline. The other measures can usually be specified as a function of LOC.

In knowledge engineering we also need a concrete baseline to use in the construction of other size-dependent metrics. We considered several options, the following two being the most promising:

• *The total number of CODE4 concepts*, M_{TOT}: This very physical size measure is appealing as an analog to lines of code, with which it shares the problem of being poorly correlated with complexity and time-to-create. Unlike lines of code however, CODE4 concepts are rather diverse in their nature, and many are created without typical users realizing it (e.g. a 'term' concept is automatically added when a user types a new name for another concept). To combat this difference in importance, we considered constructing a metric that gives different weights to each class of concept (types, terms etc.), but abandoned that approach due to lack of data.

M_{TOT} is useful as a measure of how much memory and loading-time a knowledge base requires, and correlates well with the response time of CODE4's user interface.

The number of main subjects, M_{MSUBJ}: This count excludes concepts about which nothing is said (e.g. example instances) as well as CODE4's 'special' concepts: properties, statements, metaconcepts and terms. It ignores how much detail has been filled in for each concept; this makes it uncorrelated with non-size aspects of complexity and thus more useful than M_{TOT}. Other advantages are that people have an intuitive sense of what M_{MSUBJ} means, and can estimate it early in the development of a knowledge base: It is common for people early in development to roughly categorize and name the important main subjects in their domain, and then work on adding detail.

Both M_{TOT} and M_{MSUBJ} have the problem that concepts differ in importance. For example, in many CODE4 knowledge bases people create a core of concepts which are central to their domain and about which they add much detail. Typically though, users also add a significant number (10-30%) of concepts that are outside or peripheral to the domain. In a typical case, a user creating a zoology knowledge base might sketch out a rough hierarchy of plants.

We settled on M_{MSUBJ} as the most useful basic measure of size due to its ability to be estimated early and its non-size complexity independence. In our user study this metric ranged from 30 to 278 with a mean of 107 and a standard deviation of 61. We have found that knowledge bases in these ranges are relatively easy to browse and keep consistent and correct. When the number of main subjects rises over 500, even CODE4's powerful browsing capabilities are taxed.

Independent Measures of Complexity: A number of factors contribute to the complexity of a knowledge base. We have derived seven complexity metrics that are independent of the raw size. We have also attempted to make these as independent of each other as possible. Each of these complexity measures falls in the range of 0 to 1 so that they can be easily visualized as percentages, and so they can be easily combined to form compound metrics.

• *Userprop Ratio*, M_{UPMS}: This is the square of the ratio of user (non-primitive) properties to the sum of user properties and main subjects. If a user adds no properties, then M_{UPMS} is zero; M_{UPMS} approaches one as large numbers of properties are added to a knowledge

base. In the average knowledge base, the number of user properties tends to approximately equal the number of main subjects, hence M_{UPMS} averages 0.3 (0.54 squared, i.e. people seem to add just over one new property for every main subject).

A possible flaw with this metric is that it is not open-ended: For each additional property, the increase in the metric is less. Thus in a 100 main-subject knowledge base, increasing the number of user properties from zero to 100 causes a 0.25 increase, whereas raising it another 100 only causes a 0.19 increase. If the metric were not squared, this problem would be much worse (the first 100 would cause a 0.5 increase and the next 100 only a 0.17 increase). In practice these diminishing returns only become severe at numbers of properties well above what we have encountered, and intuitively there appears to be diminishing returns in the subjective sense of complexity as well.

• *Detail*, M_{DET}: This metric gives the fraction of inherited properties that are locally modified. If no statement values are filled in at all, this metric is zero. If every possible property inheriting to a main subject is given a specialized value, then M_{DET} equals one. As can be seen from table 2, M_{DET} never exceeds 0.5 in practice – in fact a value near one indicates that inheritance is not being properly taken advantage of, and there is thus no knowledge reuse with its complexity-reducing effect.

Whereas M_{UPMS} measures the *potential* number of specialized statements , M_{DET} indicates the amount of *use* of that potential. The next measure, M_{SFORM}, goes one step further and measures to what degree that use involves a systematic syntax.

• *Statement Formality*, M_{SFORM}: This measures the fraction of statement values on main subjects that are *interpretable* by CODE4 so that they become links to other concepts in the knowledge base. If M_{SFORM} is zero, then the user has merely placed arbitrary expressions in all values. The higher M_{SFORM}, the more CODE4 is able to infer additional network structures (i.e. 'knowledge maps' such as the part-of relation) inherent in the knowledge.

• *Diversity*, M_{DIV}: While a high measure of properties, detail and formality may indicate that substantial work has been done to describe each main subject, that detail may be largely in the form of very subtle differences. There may be a large amount of what really amounts to mere data, expressed as statements of the same set of properties about each main subject. For example in a knowledge base describing categories of cars, hundreds of cars may be described but only using a fixed set of properties (engine size, fuel consumption etc.) – we subjectively judge such a knowledge base to be rather simple despite having a lot of 'facts'; M_{DIV} attempts to quantify this subjective feeling.

Our diversity metric measures the degree to which the introduction of properties is evenly spread among main subjects. If all properties are introduced in one place (e.g. at the top concept) then M_{DIV} is close to zero because the knowledge base is be judged to be simpler. Likewise, M_{DIV} is close to zero if properties are introduced mostly on leaf concepts (so there is no inheritance). Maximum M_{DIV} complexity of one is achieved when some properties are introduced at the top of the isa hierarchy, some in the middle and some at the leaves. At table 2 shows, among complexity metrics M_{DIV} has the greatest degree of variability in the knowledge bases created by the subjects in our study.

• *Second Order Knowledge*, M_{SOK}: This measures the use of synonyms and metaconcepts, indicating to what degree a knowledge base contains an added layer of complexity: I.e. the extent to which users have described the *concepts themselves* (in addition to the things represented by the concept) and have paid attention to linguistic issues. M_{SOK} is zero if neither of these features is used, and is one if every main subject has both multiple terms and a metaconcept.

• *Isa complexity,* M_{ISA}: This metric combines two factors in order to measure the complexity of the inheritance hierarchy. The first factor is the ratio of leaf types to non-leaf types (ignoring instance concepts which are always leaves). This factor is a simple function of the branching factor, however we choose not to use the latter because it is open ended. The leaf type ratio is 0.5 when the isa hierarchy is a binary tree. It would approach zero in the ridiculous case of a unary 'tree' with just a single superconcept-subconcept chain and it would approach one if every concept were an immediate subconcept of the top concept. On its own, the leaf type ratio has the undesirable property that for a very shallow hierarchy (e.g. just two levels) with a high branching factor it gives a reading that is unreasonably high, from a subjective standpoint.

To correct this problem with leaf type ratio, we factor into M_{ISA} the average number of concepts inherited by each main subject. This second factor is related to hierarchy depth but depends to some extent on the amount of multiple inheritance.

M_{ISA} is calculated using the following formula:

$$M_{ISA} = \text{leaf-type-ratio} - (\text{leaf-type-ratio} / \text{average-concepts-inherited})$$

M_{ISA} is zero in either of the following cases: 1) when there is just a single layer of concepts below the top concept (no matter how many concepts); or 2) when the branching factor is one. M_{ISA} approaches the value of the leaf type ratio (e.g. 0.5 for a binary tree) as the tree gets deeper (as the average number of direct or indirect parents per main subject increases).

• *Multiple Inheritance,* M_{MI}: This measures the extent to which main subjects have more than one parent, thus introducing complex issues associated with multiple inheritance such as teh combinationb of values when two inherited values conflict. If just single inheritance is present, this metric is zero. M_{MI} actually records the ratio of *extra* parents to main subjects, thus if all main subjects had two parents (impossible in fact because the ones near the top cannot have more than one parent) then the metric would be one; it could also be one if a substantial number of parents have more than two parents. Although as described the metric could theoretically read above one, we believe that this could not be the case in any reasonable knowledge base. Thus we impose a limit of one that the metric cannot exceed, even if there are an excessive number of parents.

As mentioned at the beginning of this subsection, we developed the above metrics with the hypothesis that they are independent of each other. As table 3 indicates, for the most part we have been successful. The only notable exception is the negative correlation between the isa complexity and the amount of multiple inheritance. This can be accounted for theoretically because if there are more parent concepts to be multiply inherited (including by leaves), then the proportion of leaf types should decrease. Despite this moderate correlation, we think that having a separate measure of multiple inheritance is useful.

	Multiple Inher. M_{MI}	Isa Complex. M_{ISA}	Second Order M_{SOK}	Diversity M_{DIV}	Statement Formality M_{SFORM}	Detail M_{DET}
Userprop ratio: M_{UPMS}	-0.05	-0.16	0.24	0.08	-0.17	0.20
Detail: M_{DET}	0.05	-0.10	-0.16	-0.11	-0.17	
Statement Formality: M_{SFORM}	-0.16	0.17	-0.05	0.27		
Diversity: M_{DIV}	0.01	0.15	-0.14			
Second Order Knowledge: M_{SOK}	-0.13	-0.04				
Isa complexity: M_{ISA}	-0.46					

Table 3: Coefficients of linear correlation among the six complexity metrics

Compound Measures of Complexity: We combine the above simple metrics into compound metrics that give useful overall pictures of a knowledge base. For our initial study we weight all combined metrics equally because we do not yet have enough data to justify an unequal weighting.

• *Apparent completeness*, M_{ACPLT}: For this metric we combine those metrics that, intuitively, should steadily increase as a project progresses. We also seek to create a metric that ranges between zero and 100 percent so that users can obtain an intuitive impression of how much work needs to be done on a knowledge base.

The metrics composing M_{ACPLT} are: M_{UPMS}, indicating the the extent to which properties have been added; M_{DET}, indicating the proportion of statements filled in, and M_{SFORM} indicating the extent to which the knowledge base has been formalized. We considered adding M_{SOK}, but decided against it because we believe the amount of second order knowledge may vary too significantly among domains.

Taking the unadjusted average of the component metrics would result in an arbitrary bias in favour of metrics that have higher expected values. We correct for this as follows: before averaging, we normalize each metric by multiplying it by a factor that results in the *maximum* reading for the data of our study being one. We use the maximum because we want knowledge bases with the highest readings for each component metric to appear close to 100% complete. We also want to have M_{ACPLT} be zero if all its component metrics are zero.

The derived formula for M_{ACPLT} is thus as follows:

$$M_{ACPLT} = (1.4\ M_{UPMS} + 2.2\ M_{DET} + 2\ M_{SFORM}) / 3$$

As can be seen from table 2, the knowledge bases in our study have a wide range of completeness. Using the questionnaire, we asked people to rate the completeness of their knowledge bases; unfortunately we discovered later that this particular question was interpreted in several different ways so we as of yet have insufficient evidence for the validity of the above metric. The strongest apparent weakness is that statement formality may be weighted too highly – many people appear highly satisfied with very informal knowledge bases. In order to adjust the weightings in the formula, we would adjust both the constant factors and the denominator in the equation so the result still franges from zero to 100%.

An interesting derivative use of M_{ACPLT} would be to apply it to different subhierarchies of a knowledge base in order to determine which areas need work, or alternatively, which areas contain more useful knowledge.

• *Pure complexity*, M_{PCPLX}: To compute this metric, we combine all the independent complexity measures, again without weighting. Our objective is to give a size-independent measure of how 'difficult' or 'sophisticated' a knowledge base is. M_{PCPLX} can be seen as similar to the complexity multipliers in the function point or COCOMO methods.

We seek to create a metric where above-average complexity results in a value greater than one and below-average complexity yields a value less than one. To do this we first add a constant to each metric to bring its average up to one, and then multiply the results together.

The derived formula for M_{PCPLX} is thus as follows:

$$M_{PCPLX} = (M_{UPMS} + 0.7) * (M_{DET} + 0.86) * (M_{SFORM} + 0.84) * (M_{DIV} + 0.27) * (M_{SOK} + 0.92) * (M_{ISA} + 0.57) * (M_{MI} + 0.87)$$

As expected, the average of M_{PCPLX} for the knowledge bases in our study is one. The complexities vary over a wide range, yielding useful information about the knowledge bases, independently of size.

• *Overall complexity*, M_{OCPLX}: Our overall complexity measure is simply M_{MSUBJ} multiplied by M_{PCPLX}. In other words we adjust the count of the number of main subjects using our measure of 'pure' complexity of the knowledge base. M_{OCPLX} is intended to serve as a measure of productivity or information content.

We asked the participants how many hours they spent creating each knowledge base. We then computed the coefficients of correlation between this estimate and our three size metrics (the basic counts M_{TOT} and M_{MSUBJ}, plus our computed M_{OCPLX}) in order to see how well each could act as a productivity measure. As table 4 shows, M_{OCPLX} scored higher than the others although the correlation was far from perfect. Some reasons for lack off correlation are: a) the users differed in skill level (some were beginners and others were experts); b) the time estimates were subjective because the users spent time over many weeks and it was hard for them to account for all of their time, and c) it was hard for them to segregate time spent using CODE4 from time spent on research away from the system.

Metric	**Correlation with estimated time**	
M_{TOT}	0.49	Total number of concepts
M_{MSUBJ}	0.36	Number of main subjects
M_{OCPLX}	0.55	Overall complexity

Table 4: Correlations between metrics and time to build knowledge bases

As an experiment we translated a substantial portion of Bruce Porter's 'Botany Knowledge Base' (Acker 92) from his KM system into CODE4-KR, and tried

manipulating it. With an M_{OCPLX} of 836 (M_{MSUBJ} = 768; M_{TOT} = 6178) it forced us to add new features to CODE4 to permit more control of the segregation of sub-portions of the knowledge base for study. Although CODE4's user interface is capable of opening a substantially wider 'window' on s knowledge base than is Porter's KM system, CODE4 requires commensurately more skilled manipulation to do this (when browsing smaller knowledge bases, CODE4 requires substantially less skill to operate than KM).

4. SUBJECTIVE MEASURES OF CODE4'S FEATURES

Using the questionnaire, subjects were asked what aspects or 'features' of CODE4-KR (CODE4's knowledge representation) they feel were important in representing or conveying the knowledge in their *particular* knowledge bases, and to what degrees were these aspects important. As a result of this study, nine distinct aspects of the representation were identified. These are listed and categorized in table 5. Table 5 also shows the rated importance of each aspect on a scale of 0 to 10.

In table 5, we have divided the aspects into three categories: Structural aspects of representation deal with interconnection and positioning of nodes, considering the knowledge base to be a semantic net. Terminological aspects deal with the nodes themselves and how they are labelled. Statement aspects deal with how the nodes are described in detail. These aspects are discussed in further detail in the next few sections; to understand more, the reader is urged to consult (Skuce and Lethbridge, 1994).

One of our most significant hypotheses when designing CODE4 has been the need to allow for users to choose their level of informality[3]. By this we mean that users usually want to sketch knowledge in a rough form during the early stages of development, and then gradually alter it by modifying both structure and syntax so that they conform to standards, correctly convey the intended meaning, and can be used by inference mechanisms. This implies designing a knowledge representation that tightly integrates formal and informal aspects. (Lethbridge, 1991) and (Lethbridge and Skuce, 1992a and 1992b) describe various aspects of the CODE4 design that relate to the formality spectrum. (Shipman and Marshall, 1993) also make strong arguments for a formality spectrum.

In table 5, we have indicated which aspects are oriented towards informal use of CODE4, and which features are for more formal use.

Aspect	Importance
Structural aspects (section 4.1)	
Conceptual structure, i.e. the 'knowledge maps' in a CODE4 KB (formal)	9.5

[3] There has been considerable debate about our use of the terms 'formality' and 'informality'. Few people object to us using the term 'informal' to describe the use of CODE4 to roughly sketch knowledge, without being constrained to rigidly defined syntaxes. However some object to us using the term 'formal' for the inverse, because they prefer to preserve that word only for representations with mathematically defined semantics. We assert, however, that our use of 'formal' corresponds with dictionary definitions and conventional non-computer-science use; and furthermore we find no better word that captures what we are trying to express.

Presentation structure (informal)	
Layout (in three dimensions)	5.4
Order (in two dimensions)	2.6
Terminological aspects (section 4.2)	
Names (formal or informal)	
Main subject names	9.6
Property (relation) names	9.9
2nd order relation names (dimensions)	6.2
Graphic symbols (informal)	3.3
Statement aspects (section 4.3)	
Formal statements	5.1
Informal statements	7.9

Table 5: Subjects' perceptions about the importance of certain aspects of the knowledge representation to the conveyance of the knowledge in their knowledge bases.

4.1 Structural Aspects of CODE4

Many systems are capable of automatically generating graphs from a description of the topography of a knowledge base (using a particular hierarchy or 'knowledge map' – say isa, or part-of). We discovered early in our use of CODE2, however, that users like to rearrange their graphs – the graphs have more *meaning* to them when they are rearranged. A possible conclusion from this is that the graph-drawing algorithms are not optimal. That is certainly part of the truth, but we have also concluded that people are actually representing implicit knowledge when they create their own special layouts, or when they merely choose to *order* subconcepts in a certain way.

CODE4 has features that allow users to manually lay out graphs and then to save them for recall. Users can save *multiple* layouts for the same information, and can combine layouts of subgraphs in various ways. Similar capabilities are also available for non-graphical user interface components (e.g. outline browsers, where the ordering of elements under a particular parent can be manipulated).

Because the knowledge implicit in ordering or layouts only has meaning to the users, and cannot be interpreted by inference mechanisms, we consider it informal. We use the term *structural formality* to refer to the degree to which the decisions about the spatial positioning of concepts are purely based on the topography of the graph being represented.

4.2 Terminological Aspects

We use the term *terminological formality* to refer to the degree to which a system enforces a strict rule of one constrained-syntax identifier per concept. Most systems do just that, requiring the user to pick a unique term using alphanumeric characters. Such systems require these unique identifiers to distinguish concepts.

Based on lessons from CODE2, where choosing terms was found to be one of the most difficult tasks for users, we choose to clearly separate the 'uniqueness' role from the

'identifier' role of terms. Looked at another way, we choose to separate the task of identifying a concept to *CODE4* from identifying a concept to the *users* of CODE4. CODE4 maintains automatically-generated internal identifiers, but users normally do not see these. When a concept is first created, the system invents a label for it, often merely based on its location in the inheritance hierarchy. For example, if a user adds a new subconcept of 'horse' the default term will be 'specialized horse'. Indeed if two or more new subconcepts of horse are added, they will all be called 'specialized horse'. The user is responsible for knowing which is which, perhaps based on the order in which they are listed or the properties they have. Presumably the user will give such concepts meaningful names, but the burden of doing this at the instant of creation is lessened.

In addition to allowing variable numbers of terms per concept, CODE4 also does not constrain the syntax of terms. Any ASCII sequence can be used, or, alternately, one or more bit-mapped graphics can be used. In future we intend to extend this idea so that concepts can be identified by particular fonts, shapes and colours.

The subjects in our study reported that for 31% of concepts, was no one standard term they could think of. This figure ranged from 10% to 90% depending on knowledge base (std. dev. 25%). To combat this problem, subjects had four options available:

1) Invent a new term.
2) Use a single term to refer to more than one concept.
3) Use more than one term for a concept.
4) Use no term (leave the possibly non-unique default label)

For 19% of all concepts, the users took the first option. There were 1.04 terms per concept in the average knowledge base indicating significant use of option 3.

4.3 Statement Aspects

The subjects were asked to estimate the formality of the statements[4] in their knowledge bases, using the following criterion: "Consider knowledge more formal if it has a high proportion of property values with actual links to other concepts rather than just arbitrary [unparsable] expressions".

Estimates of formality ranged from 0% to 90%. The mean was 54% and the standard deviation was 26%. The actual formality of first-order knowledge in the knowledge bases was then computed. The actual formality ranged from 0% to 48% with a mean of 16% (see M_{SFORM} in table 2). The coefficient of correlation between the subjects' estimates and the measured formality was 0.27 which, although positive, indicates that the subjects either have an unclear idea of what formality means or cannot easily estimate it. Both reasons are likely true.

When asked about which type of statements were most important in conveying its content the subjects ranked formal statements significantly lower than informal ones (51% vs. 79%). (Although see table 6 for somewhat contradictory data) When using informal state-

[4] A statement corresponds to the occurrence of a property in a subject concept. They are called concept-slot-value triples in KM, for example. A statement has a 'value' and possibly other facets.

useful to you think the features might be in *general?''*. For both questions the subjects used a scale where -5 means 'very harmful', zero means of no use and 5 means essential. We combined the scores for the two questions to produce numbers on a scale of -10 to 10. All the mean scores are positive.

Table 6 shows the subjects' ratings of 12 knowledge representation features for which we obtained statistically significant results. Although several features involving informality have slightly lower ratings than formal features, this does not contradict our hypothesis about the importance of informality because *all* the features listed were judged to be important.

Feature	**Rating**	**Type of feature**
Property hierarchy *	10	Structural
Inheritance	9.7	Inference
Multiple inheritance	9.7	Inference
Multiple property parents *	9.4	Structural
Modality facet	8.6	Statement
Facets in general	8.5	Statement
Formal references	8	Statement
Delegation	7.7	Inference
Metaconcepts *	7.5	Structural / statement
Flexibility in adding facets *	7.4	Structural / statement
Informality in statement values *	7.3	Statement (informal)
Synonyms *	7.3	Terminological
Ability to use any string in a term *	7.1	Terminological (informal)

The built-in facets	7	Structural / statement

Table 6: Ratings (in a scale of -10 to 10) of all the knowledge representation features for which statistically significant results were obtained. A single asterisk indicates the feature is rarely found with similar capabilities in comparable systems.

4.6 Suitability of CODE4 to Tasks

In general there was a high satisfaction with the use of CODE4. In almost all cases, the subjects reported that they obtained substantial insight about the subject matter when using CODE4. On a scale where 0 indicates that constructing the knowledge base results in no new insights and 10 indicates that many new insights were obtained, the mean response was 7.7 (std. dev. = 2.5).

The subjects were then asked to compare CODE4 with other representational technologies with which they were familiar. They were asked, only for those technologies with which they were familiar, "How easily could you have represented the same knowledge using the following types of software or representation methods?". The answers were placed on a scale where -5 means that the knowledge can be represented much *more* easily using the alternate technology than using CODE4, zero means that CODE4 equals the technology in representational ease, and 5 means that the using the alternate technology instead of CODE4 would have resulted in much more difficulty.

In all cases where there was sufficient data, the subjects ranked CODE4 ahead of the alternate technologies. Figure 3 summarizes the results. In particular it is notable that CODE4 is perceived as more useful than both informal representation techniques (e.g. natural language) and more formal representation techniques (e.g. predicate calculus and conceptual graphs).

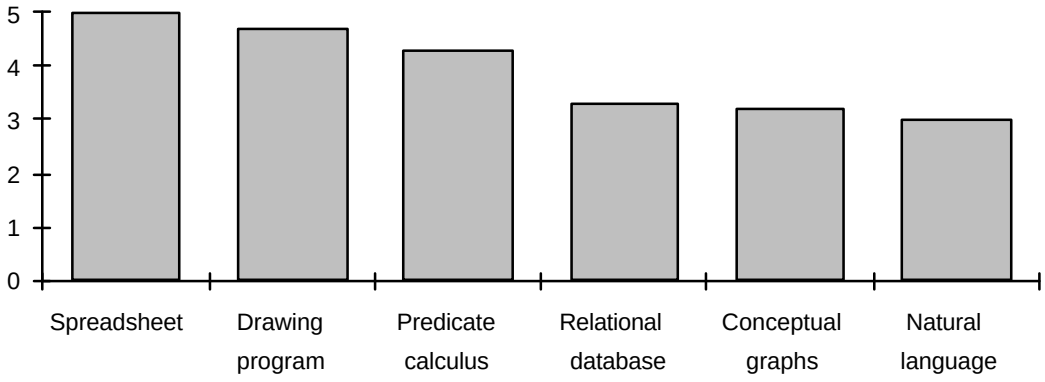

Figure 3: Perceived difficulty of use of representational technologies. Larger numbers indicate that the technology is perceived as being more difficult to use for the kind of representational tasks studied. CODE4 (the baseline for comparison) is zero. A negative number (of which there are none) would indicate that the technology is perceived as easier than CODE4.

The subjects were also asked to compare CODE4 to technologies such as KIF, Ontolingua, KL-One derivatives, hypertext and outline processors. Unfortunately there

was insufficient experience among subjects for statistically significant conclusions about these technologies.

5. CONCLUSIONS AND FUTURE WORK

Detailed studies of knowledge acquisition cases can yield useful insights. We analysed the use of CODE4 using subjective questionnaires as well as objective computations performed on the resulting knowledge bases. In our study, eight users spent about 2000 hours creating 18 knowledge bases.

The following conclusions have resulted from the above process:

1) The ability to measure characteristics of knowledge bases can help with several tasks including measuring productivity, determining how 'complete' knowledge is and predicting how long it will take to develop a knowledge base. The metrics discussed in this paper have proved useful in our CODE4 research, and could readily be calculated for other types of knowledge representation system.

2) There appear to be at least six different ways of characterizing the complexity of a knowledge base that are independent of size and of each other. These include the relative number of properties, the extent to which concepts are described using the properties, the formality of statements using the properties, the degree to which properties are introduced in an evenly-distributed manner, the proportion of second-order knowledge and the structural complexity of the isa hierarchy. These, plus measures of raw size, can be combined to create useful compound metrics.

3) A knowledge management system that is capable of allowing users to represent and manipulate informal knowledge is indeed useful. CODE4 users appear to value the combination of a knowledge based system's capabilities, with the freedom to express ideas using a variety of textual and graphic syntaxes and terminologies.

4) Most systems or languages are not designed to allow a spectrum of formality, therefore such systems are restricted in their usefulness when it comes to the task of helping users structure their thoughts. While highly formal systems may be useful for the creation of knowledge bases for inference-intensive automated tasks, such systems have substantially less ability to act as an amplifier of human thought processes.

5) Several important aspects of informal knowledge representation cannot be adequately exploited without the presence of a graphical user interface. Of particular importance is the ability to manipulate graphs or textual hierarchies showing structure. While knowledge acquisition systems which only focus on formal knowledge for inference systems may find such a user interface useful; it becomes *essential* for systems that are designed to help people explore and structure their thoughts. The design of such a user interface must be done in tandem with the design of the knowledge representation.

We see several avenues of future research. Of significant use would be to gather more user data in order to improve, fine-tune and validate the metrics. Another way to fine-tune and validate the metrics would be to have several CODE4 experts take the knowledge bases created during our study, and judge various aspects of their complexity. The results of these judgements would be compared with the metrics.

Skuce, D.,Wang, S., & Beauvillé, Y. (1989). “A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis”. Banff KAW, 89