To appear: International Journal of Human-Computer Studies, 1998, 49, pages 867-893

Towards Situated Knowledge Acquisition

Tim Menzies
NASA/WVU Software Research Lab, 100 University Drive, Fairmont, WV USA
tim@menzies.com

Abstract

Situated cognition is not a mere philosophical concern: it has pragmatic implications for current practice in knowledge acquisition. Tools must move from being design-focused to being maintenance-focused. Reuse-based approaches (e.g. using problem solving methods) will fail unless the reused descriptions can be extensively modified to suit the new situation. Knowledge engineers must model not only descriptions of expert knowledge, but also the environment in which a knowledge base will perform. Descriptions of knowledge must be constantly re-evaluated. This re-evaluation process has implications for assessing representations[1].

Introduction

Consider a knowledge base which is conceived, built, and used. How can we improve our skills for developing such knowledge bases?

- If most of the changes to that knowledge base occur before its usage, then we would look to optimising the design process: i.e. conception to construction. Standard knowledge acquisition practice (e.g. KADS (Wielinga, Schreiber, & Breuker 1992; Breuker & de Velde (eds) 1994)) is very focused on optimising design.
- If most of the changes to that knowledge base occur once it is being used, and we should look to optimising the maintenance process: usage to reconception to reconstruction. Very few knowledge acquisition tools are maintenance-focused (exception: ripple-down-rules (Compton & Jansen 1990)).

The claim of this article is that is if we accept situated cognition, then knowledge acquisition must move away from current approaches which seek to optimise design. That is, situated cognition challenges established current design-focused knowledge acquisition practice.

It could be argued that it is a false dichotomy to separate an emphasis on design from an emphasis on maintenance. Isn't good design the best means of simplifying maintenance? Perhaps not. As we shall see below, certain successful maintenance-focused tools can take a very minimal approach to initial design.

[1]This work was performed while the author was located at the Artificial Intelligence Department, School of Computer Science and Engineering, University of NSW, Australia, 2052

It could also be argued that a philosophical perspective on human reasoning has little relevance for tool builders such as pragmatic knowledge engineers. Situated cognition seems a notion of science, 'what is the basis of natural human behavior', while knowledge acquisition is a notion of engineering, 'how can we build cost-effective systems'. However, a knowledge base is an embodiment of some scientific model of the world. We build models to explicate and share our understanding of a domain. Idiosyncrasies in our conceptualisation process should be managed in our modeling tools. This article focuses on the idiosyncrasy of *changing our minds*. People, even experts, change their mind frequently. Due to situated cognition, a description of some treasured belief from today may be different than yesterday's description of that belief. Will this difference significantly disturb the modeling process? Later in this article, we will apply these two tests to check if situated cognition is a pragmatic concern for tool builders:

- *Test 1*: Design-focused knowledge acquisition techniques assume that abstracted portions of old knowledge bases (ontologies (Gruber 1993) or problem solving methods (Breuker & de Velde (eds) 1994; Chandrasekaran, Johnson, & Smith 1992)) can be reused for new applications. Situated cognition is a major concern for current practice if knowledge base changes preclude the successful application of these abstracted portions of old knowledge bases to new situations.
- *Test 2*: Situated cognition claims that we should expect descriptions of knowledge (e.g. an ascii knowledge base) to undergo major change during its lifetime. Situated cognition is a pragmatic concern if knowledge base change is a major issue for real-world knowledge bases.

Vera and Simon offer other arguments against changing current practice due to situated cognition (Vera & Simon 1993b; 1993a; 1993c). Firstly, they argue that the physical symbol system hypothesis (Newell & Simon 1972) has been a fruitful paradigm which can reproduce many known behaviours of experts. This is a compelling argument. Why should we change current practice when current practice has produced so many successful expert systems (e.g., PROSPECTOR (Campbell *et al.* 1982; Duda, Hart, & Reboh 1985), XCON (Bachant & McDermott 1984), VT (Marcus, Stout, & McDermott 1987), PIGE (Menzies *et* *al.* 1992))? Secondly, Vera and Simon decline to reject the physical symbol system hypothesis for situated cognition since, if they adopted, e.g. Clancey's situated paradigm (Clancey 1987), then Vera and Simon are unclear on what predictions can be made and what experiments can be performed. That is, Vera and Simon argue that situated cognition is un-falsifiable and unscientific.

This article will present arguments to reject the Vera and Simon view. Firstly, certain central claims central to (e.g.) Simon's position are still unresolved research issues (see the discussion below on expertise transfer). Secondly, Vera and Simon comment on the successes of working descriptions of human knowledge, not on the effort involved in constructing or preserving those descriptions. Many researchers have recognised that our design concepts for knowledge-based systems were incomplete (Buchanan & Smith 1989). For example, Steels (Steels 1994) cites an example where an expert could not solve a problem over the phone but, as soon as they walked into the room where the trouble was, could solve it instantly. Examples like this suggest that we have undervalued the impact of situation-specific reasoning. Thirdly, Vera and Simon are only arguing against a particular extreme type of situated cognition. We can adopt *weak situated cognition* (defined below) and still retain the physical symbol system hypothesis, but only if we can demonstrate that we can manage changes to our descriptions of knowledge. Fourthly, it is not true that situated cognition is unfalsifiable and unscientific. Two experiments with situated cognition were offered above (the two tests for change and reuse). The end of this article offers a third experiment with a falsifiable hypothesis on situated knowledge engineering.

The rest of this article is structured as follows. After a brief review of the situated cognition literature, the two tests defined above for change and reuse are applied. In the literature we will find evidence that (i) large scale change may be common and (ii) successful reuse is rare. Hence, we have some motivation for discussing alternatives to current knowledge engineering practice. Four general principles for situated knowledge engineering will then be presented: modeling the environment of a knowledge-based system; an emphasis on maintenance rather than design; knowledge engineering metrics; and representation options which enable the continual testing of theories. Experiments with testable representations will be described.

A Brief Review of Situated Cognition

This section reviews some of the literature that lead to the situated cognition stance. For more on this material, see the other articles in this issue.

Proponents of situated cognition argue many diverse views. These can be roughly divided into three camps: the situated premise, weak situated cognition, and strong situated cognition.

The Situated Premise

The situated premise argues that human cognition cannot be accurately modeled by context-independent assertions (the so-called logical-AI approach critiqued by Birnbaum (Birnbaum 1991), amongst others). Dreyfus argues that the context-dependent nature of human knowledge makes it fundamentally impossible to reproduce in symbolic descriptions (Dreyfus 1979). Searle takes a similar stand, claiming that the only device that can replicate human intelligence is another human (Searle 1980; 1982; 1995) since only humans can share the same context.

Weak Situated Cognition

Weak situated cognition argues that the reason for the situated premise is that when a human agent uses a description of knowledge, they continually reinterpret that description in the context of the current problem. Clancey (Clancey 1987; 1991), Winograd and Flores (Winograd & Flores 1987) argue that it is a mistake to confuse the descriptions which humans use to co-ordinate their activities and reflect about their actions (i.e. language) with how humans might generate their minute-to-minute behaviour. In this view, human inference is not just matching and retrieving old descriptions. Rather, these structures are created on-the-fly:

> Every action is an interpretation of the current situation, based on the entire history of our interactions. In some sense every action is automatically an inductive, *adjusted* process. (Clancey 1987),p238.... The neural structures and processes that coordinate perception and action are created during activity, *not* retrieved and rotely applied, merely reconstructed, or calculated via stored rules and pattern descriptions (Clancey 1993),p94.

According to Clancey (personal communication) memory is taken to be a reconstructive act: bringing knowledge to bear is really a constructive act that (re)creates knowledge in light of the current situation. Using descriptive models (in the world or imagination) involves this reconceptualization, so knowledge (in the brain) changes. Hence our descriptions of knowledge should not be treated as context-independent assertions. Models are only descriptions of the world, human behavior, etc. and as descriptions they are susceptible to breakdown when the environment changes or human intentions or values change.

Note that weak situated cognition does not imply a rejection of the symbolic modeling paradigm. Models are are not useless. However, they are perhaps best viewed as tools for facilitating a discussion about a domain. Such models allow us to plan about the future and reflecting on action rather than immediately reacting to a new situation:

> Human reasoning is immensely more successful by our ability to simulate what might happen, to visualize possible outcomes and prepare for them. We do this by reflecting, saying what we expect, and responding to what we say (Clancey 1987).

However, Clancey's descriptions of knowledge are not as fixed as those in standard knowledge engineering:

> It remains to explain how (the knowledge descriptions) *develop*... Most learning programs grammatically describe how representations accumulate within a fixed language. They don't explain how representations are

created, or more generally, the evolution of new routines not described by the given grammar (Clancey 1991),p279.

Note that weak situated cognition does not necessarily imply that knowledge engineers will be continually revising the content knowledge bases. For example, the knowledge base may remain static while the interpretation placed on the output of the system may change over time (e.g. certain portions of the output reports are routinely ignored). However, if operational characteristics of the working program are ever used to revise the conceptual model, weak situated cognition will influence the nature of those revisions. For example:

- Many real-world complex domains can only be understood via an exploratory methodology. For example, software solutions to many standard business problems require extensive iterative development (e.g. (Jacobson *et al.* 1992)). In such exploratory approaches, interpretations of the behaviour of a running system informs revisions to that system.
- Experience can result in new insights. Often, only after a working model breaks down, do we realise some key missing knowledge to be added to the knowledge base (Fischer *et al.* 1996; Compton & Jansen 1990). Shalin et.al. (Shalin *et al.* 1997) argue that experts routinely modify records of *accepted practice* after a situation-specific assessment of the utility of old accepted practice (this study is discussed below). Note that the added knowledge will be an interpretation of watching old knowledge exercise in some new situation.
- Researchers into decision support tools (Bradshaw, Ford, & Adams-Webber 1991; Phillips 1984) argue that model construction is a communal process that generates descriptions which explicate a community's understanding of a problem. If the community changes, then the explicit record of the communities shared understanding also changes. Such an explicit expression of current beliefs may prompt further investigation and model revision. The very act of using models leads people to reinterpret what they mean and to adapt them to new situations; i.e. writing down and exercising descriptions of knowledge can lead to modifications of those descriptions of knowledge.

Strong Situated Cognition

Strong situated cognition goes one step further than weak situated cognition. The influence of context is so great, says strong situated cognition, that we should use pure reactive systems that interact directly with the context without first reflecting over some description (e.g. (Brooks 1991)). Birnbaum (Birnbaum 1991) and McDermott (McDermott 1987) argue that the obvious alternative to logical AI is some type of procedural/functional semantics.

Opponents of the situated view (e.g. Vera and Simon) often are opposing strong situated cognition since it implies a total rejection of the symbolic modeling paradigm. This is a very drastic move since much of our understanding of human expertise is based around symbolic models. An often-repeated observation is that, when asked to explain their expertise, experts produce a richer and more abstract description than novices. For example:

- Explanations of disease processes from final year medical students contain more intermediary concepts than first year students. Further, the explanations from first year students showed that these novices focus on the surface features (initial observations) of a problem (Patel & Ramoni 1997).
- Anderson reports similar results in the fields of physics, mathematics, computer programming, and medical diagnosis (Anderson 1990), chp.9. Novices focus on the surface features of a problem. Experts abstract from the particulars of a problem. In the abstracted form, experts use a description language which can offer uniform principles across many problems.

One possible conclusion from these studies is that experts are experts because they use a richer internal store of symbolic beliefs. The whole notion of a knowledge-based system is based around the representation and (re)interpretation of such symbols. If we reject this symbolic modeling, as demanded by strong situated cognition, we must reject the vast majority of our current representational toolkit.

Note however that a knowledge engineer can take a weak situated stance without rejecting symbolic modeling. In weak situated cognition, we can still model expert competency via a reflection over descriptions of expert belief. However, we must accept that these descriptions are prone to large scale change. To demonstrate that situated knowledge engineering is practical, it must be shown that change is either ignorable or manageable. The next section argues that the current literature cannot demonstrate that change is ignorable.

How Serious is the Problem of Change?

The previous section argued that large scale changes to a knowledge base are possible. This section argues that large scale change may be common.

Studies in maintenance of fielded knowledge base systems are very rare. Two such studies document the XCON and Garvan ES-1 systems. In the XCON system, half of its thousands of rules are changed every year (Soloway, Bachant, & Jensen 1987). The XCON changes can be attributed to a changing environment (XCON configured computers for DEC computers and DEC keeps realising new computers). However, even in static domains, massive change can occur. The Garvan ES-1 system (Compton *et al.* 1989) was developed in a totally static domain; i.e. the system was a post-processor to a biochemical assay unit that did not change for the lifetime of the project. In that system, maintenance never reached a logical termination point, despite years of maintenance. There was always one more major insight into the domain, one more major conceptual error, and one more significant addition (Compton *et al.* 1989). The size of the changes were quite dramatic. Rules that began as simple modular chunks of knowledge evolved into very complicated and confusing knowledge (see Figure 1).

A. Originally

```
RULE(22310.01)
IF  (bhthy or
     utsh_bhft4 or
     vhthy) and not on_t4
           and not surgery
           and (antithyroid or
                hyperthyroid)
THEN DIAGNOSIS("...thyrotoxicosis")
```

B. Same rule, 3 years later

```
RULE(22310.01)
IF  ((((T3 is missing)
       or (T3 is low and
           T3_BORD is low))
       and TSH is missing
       and vhthy
       and not (query_t4 or on_t4 or
                 or surgery or tumour
                 or antithyroid
                 or hypothyroid
                 or hyperthyroid))
      or ((((utsh_bhft4 or
            (Hythe and T4 is missing
             and TSH is missing))
            and (antithyroid or
                 hyperthyroid))
          or  utsh_bhft4
          or  ((Hythe or borthy)
                and T3 is missing
                and (TSH is undetect
                     or TSH is low)))
          and not on_t4 and not
              (tumour or surgery)))
     and (TT4 isnt low or T4U isnt low)
THEN DIAGNOSIS("...thyrotoxicosis")
```

Figure 1: A rule maintained for 3 years. From (Compton *et al.* 1989).

Are the large scale changes seen in XCON and Garvan ES-1 the usual situation for a knowledge base system? Given the poor state-of-the-art in metrics collection for knowledge based systems, we cannot say. All that can be said now is:

- We need to develop better metrics for knowledge base systems. We should routinely measure what percentage of a knowledge base is changed over its life cycle. We will return to this point below in the section *Knowledge-based Metrics*.
- The available (meager) evidence is that when expert systems are studied over their lifetime, large scale changes are observed.
- The impact of these changes on current practice must be assessed. One way to make this assessment is to explore the reuse literature. If software reuse is routinely successful, then clearly some knowledge recorded symbolically in one situation can be reused in other situations. That is, the problem of change discussed above is only a minor problem. The reuse literature is explored in the next section.

Looking for Successful Reuse

Reuse implies some notion of expertise transfer; i.e. expertise recorded in one situation can be reused in another. This section discusses the expertise transfer research and concludes that direct transfer occurs in only certain unusual situations.

Recall the above discussion on symbolic models of human expertise: experts are experts because they use a richer internal store of symbolic beliefs. In this theory, learning is the process of building such sophisticated symbolic structures. These structures are abstracted away from the particulars of a domain to form structures which are reusable in different situations. This theory inspired the expertise transfer paradigm of the 1970s and 1980s (poetically described by Feigenbaum as mining the jewels in the experts head (Feigenbaum & McCorduck 1983)). The goal of expertise transfer was to find the special expert symbols and transfer them to a knowledge base. However, despite numerous attempts to document the transfer effect, it became clear that transfer was not a standard method for learning expert behaviour in a new domain. Anderson (Anderson 1990) chp.9 reports that:

- A repeated observation is that experts cannot transfer their own expertise to related areas. For example, Brazilian school children working as street vendors are expert at quickly computing change in a financial transaction. However, when exactly the same calculations were presented to the children as written numeric problems in a classroom exam, this mathematical ability degraded sharply (98 percent correct in the street, 37 percent correct in the classroom).
- Knowledge transfer would predict that novices should learn faster when experts transfer their knowledge to the novices. Yet, the dominant influence on expertise is lengthy practice with the domain and not initial training.

Experiments show that transfer is not the usual method used by experts when examining new problems. A repeated observation is that transfer only occurs when facilitated by an instructor (Reed, Ernst, & Banerji 1974;

Gick & Holyoak 1980). Hayes and Simon note that the empirical evidence for the transferability of knowledge and skills to new task situations is very mixed (Hayes & Simon 1977) (however, their study did speculate that the type of instruction was crucial to successful transfer, a speculation consistent with (Reed, Ernst, & Banerji 1974)). This requirement for instructors to facilitate transfer limits the commercial practicality of transfer-based knowledge engineering. Such instructors need to have a good understanding of both the current problem and abstracted descriptions of old problems. There are only two situations where such instructors are available:

1. Some classroom situations where students are working on small example problems that have been extensively studied by the instructor. Such situations are not relevant to commercial practice.
2. Some design situations where a senior analyst, having built some previous system, is guiding junior analysts through the construction of a system that is an exact copy of that previous system. This is a very rare commercial situation. If the new system is an exact copy of an older system, why is the new system being constructed?

Despite the poor evidence, an uncritical belief in direct expertise transfer persists. Lave is scathing in her critique of this belief:

> In sum, there is no impatience, no hint is this work (Hayes & Simon 1977; Reed, Ernst, & Banerji 1974; Gick & Holyoak 1980) that the meager evidence for transfer garnered from a very substantive body of work might indicate that the concept is seriously misconceived (Lave 1988), p39.

Studies since Lave's review offer no conclusive evidence for direct expertise transfer. These studies are examined below.

The Shalin et.al. Study

Shalin et.al. (Shalin *et al.* 1997) note that experts often constrain their behaviour using descriptions of *accepted methods* (Shalin *et al.* 1997). Communities of agents transfer descriptions of accepted practice to co-ordinate their activities. For example:

- Army patrols use a limited number of prescribed methods for navigation. Knowledge of such methods allows patrols that are not in direct communication with each other to guess how each other will react in changing circumstances.
- The social consequences of error (e.g. going to jail) encourages doctors to follow descriptions of accepted practice.
- Accepted practice is also used to anticipate the future:

> Pilots do not lower the flaps of their aircraft during take off, after sensing a lack of lift. Pilots lower the flaps of their aircraft *before takeoff*, in anticipation of well described consequences of the failure to do so (Shalin *et al.* 1997), p200.

However, Shalin et.al. note that only novices slavishly reuse accepted practice. Experts apply modified forms of accepted practice which have been extensively adapted to the current situation. Further, experts then adopt some evaluation strategy to check if the modified descriptions of accepted practice were successful. If they were not, then experts then offer revisions to the definitions of *accepted practice*. That is, in the Shalin et.al. view, direct transfer of expertise is much less important than a continually process of rapid on-the-spot adaptation. Note that:

- This Shalin et.al. view is entirely consistent with the weak situated cognition view described above.
- It will be argued at the end of this article that the need to continual assess a working theory places certain restrictions on how a theory is represented.

The Corbridge et.al. Study

This section describes the Corbridge et.al. study (Corbridge, Major, & Shadbolt 1995) which explored transfer effects in problem solving methods. Before discussing the results, we briefly review the notion of a problem solving method.

About Problem Solving Methods Much of current knowledge engineering research is focused on reusing abstractions of old designs in new situations. These abstractions are often expressed as either problem solving methods (e.g. (Breuker & de Velde (eds) 1994; Chandrasekaran, Johnson, & Smith 1992)) or ontologies (e.g. (Gruber 1993)). In the view of mainstream knowledge engineering (e.g. KADS (Wielinga, Schreiber, & Breuker 1992; Breuker & de Velde (eds) 1994)), system development becomes a structured search for an appropriate problem solving method. Once a problem solving method is found or developed, then knowledge acquisition becomes a process of filling in the details required to implement that problem solving method. For example, Clancey (Clancey 1992) offers the following problem solving method:

```
METHOD findOut (
  Uses:   subsumes/2
  Method: If an hypothesis is
          subsumed by other
          findings which are not
          present in this case
          then that hypothesis
          is wrong.
        )
```

To use this method in a knowledge base, the knowledge engineer must supply the *subsumes/2* details, e.g.

```
subsumes(surgery, neurosurgery).
subsumes(neurosurgery, recentNeurosurgery).
subsumes(recentNeurosurgery,
         ventricularUrethralShunt).
```

Mainstream knowledge engineering research argues that libraries of problem solving methods (e.g. (Benjamins 1995; Breuker & de Velde (eds) 1994; Chandrasekaran, Johnson, & Smith 1992; Motta & Zdrahal 1996; Tansley & Hayball

1993)) are a productivity tool for building a wide-variety of expert systems. All known problem solving methods (e.g. findOut, prediction, monitoring, diagnosis, cover and differentiate, propose and revise, planning, design, verification, assessment) are really combinations of a small number (say, less than 20) of reusable inference subroutines. (e.g. instantiate, generalise, abstract, specify, select, assign-value, compute, compare, match, assemble, decompose, transform). New problem solving methods can be quickly built out of these lower-level inference primitives. Domain-specific knowledge can be used in different contexts in different ways by defining different terminological mappings between domain-dependent terms (e.g. *surgery*) and the supposedly domain-independent problem solving methods.

The standard architecture for problem solving method is at least a two-layered system. In the bottom layer are domain-dependent facts in the language of the users. In the second layer are the supposedly domain-independent problem solving methods that are written in a more general language. Some mapping function is defined to connect the domain-dependent language (e.g. *surgery*) to a more general, domain-independent language (e.g. into the *subsumes/2* hierarchy). The domain-specific theory is assumed to be changeable and the same fact can take on different roles in different problem solving contexts via different mapping functions. It has hence been argued (Hoffman, Feltovich, & Ford 1997; Shadbolt & O'Hara 1997) that problem solving methods are a technology for addressing the situated knowledge issue. This may only be partially correct since while domain facts are taken to be context dependent, the problem solving methods are assumed to be reusable. That is, the problem solving method community takes a weak situated stance for domain facts but, with a few exceptions, an expertise transfer approach for problem solving methods. The exceptions are problem solving approaches that allow for the dynamic configuration of a problem solver via a depth-first traversal of a hierarchy describing problem solving options (e.g. (Benjamins & Jansweijer 1994; O'Hara & Shadbolt 1997)). This is only a partial situated stance since while the generated problem solving methods can vary according to the problem context, the background hierarchy is fixed. No guidelines are given for how experience with the running program can feedback into modifying the problem solving methods options hierarchy. Van de Velde hints at a more general mechanism in which machine learners learn model review strategies via watching human revise their models (de Velde 1993), but such work is only in its infancy.

Testing the Effectiveness of Problem Solving Methods The Corbridge et.al. (Corbridge, Major, & Shadbolt 1995) study tested the effectiveness of a problem solving method. Subjects had to extract domain facts from a transcript of a patient talking to a doctor. From the transcript, it was possible to extract 20 respiratory disorders and a total of 304 *knowledge fragments*: e.g. identification of routine tests; non-routine tests; relevant parameters; or complaints. Subjects were offered different background knowledge. One group had no model at all and a second group was given a vague, quickly written, model of diagnosis. A third group was offered the KADS problem solving method for diagnosis. This problem solving method dates back to at least 1985 (Clancey 1985) and has been extensively analysed since (e.g. (Wielinga, Schreiber, & Breuker 1992; Tansley & Hayball 1993)). A pre-experimental intuition by proponents of transfer would be that users with access to this problem solving method would recognise more diagnosis information than the other subjects. However, users of the KADS diagnosis model extracted statistically the same percentage of assertions as users of the vague model. Further, contrary to the transfer pre-experimental intuition, subjects given no background knowledge out-performed the groups with background knowledge (see Table 1).

The Corbridge study used a structural decomposition process for their KADS diagnosis model. However, according to Breuker (personal communication) medical diagnosis is usually an assessment task. Breuker argues that the poor performance of the KADS diagnosis model in the Corbridge study was due to their use of the wrong model. This view has yet to be tested empirically. In the meantime, we can say that Breuker's comments are consistent with the transfer results described above. Transfer is not a natural occurring phenomena: instructors with high-levels of skills in the use of the KADS libraries (e.g. the author of those libraries (Breuker & de Velde (eds) 1994)) is required to facilitate transfer. The scarcity of such skilled instructors limits the practicality of knowledge acquisition based on transfer.

Reuse Reports in Software Engineering

Significant levels of reuse are routinely reported in the software engineering literature. Stark reports code reuse levels of 70-80 percent using FORTRAN and some object-oriented design principles (Stark 1993). Frakes and Fox found maximum median values for reuse in requirements, design, and code reuse at 15, 70, and 40 percent respectively (Frakes & Fox 1995). If we believe these reports, then clearly some expertise recorded in old situations is being transferred to new situations. However, while these results are original and significant pieces of research, there are some drawbacks with these reports. Fenton (Fenton 1991) says that software engineering metrics should accurately report the following triad of measurements:

- Product measures: e.g. what was built and how effective was the constructed artifact.
- Resource measures: e.g. the skill level of the practioners who build the product and the time taken in the construction.
- Process measures: e.g. how an artifact was built.

Unless all three points of the Fenton-triad are reported, then it is hard to assess if (e.g.) the time taken to build a system was reduced by sacrificing the quality of the system. Reuse reports in the software engineering literature do not contain detailed process, product, and resource measures (e.g. the Stark study). Also, the Frakes and Fox study never looked at source code itself: it's data was based on a questionnaire sent and returned by post (a very poor product measure).

Model	% disorders identified	% knowledge fragments identified
Vague model	50	28
KADS model	55	34
No model	75	41

Table 1: Analysis via different models in the Corbridge study (Corbridge, Major, & Shadbolt 1995).

Reuse Reports in Knowledge Engineering

Significant levels of reuse of problem solving methods are routinely reported in the knowledge engineering literature. However, my reading of the problem solving methods literature is that these problem solving methods change more than they are reused. Between the various camps of problem solving method researchers, there is little agreement on the details of the problem solving methods. The list of primitives within the problem solving methods (e.g. findOut, select, classify, etc) from Clancey (Clancey 1992), KADS (Wielinga, Schreiber, & Breuker 1992), and the SPARK/ BURN/ FIREFIGHTER project (Marques *et al.* 1992) are significantly different. Also, the number and nature of the problem solving methods is not fixed. Often when a domain is analysed using problem solving methods, a new method is induced (Linster & Musen 1992). When we look at published problem solving methods, we see many differences. For example, (Menzies 1997) describes eight different supposedly reusable models of diagnosis (four from the problem solving method community, four from elsewhere). While some of the these views on diagnosis share some common features, they reflect fundamentally divergent different views on how to perform diagnosis. I therefore believe that, at least in the case of diagnosis, a consensus view on diagnosis has not stabilised with time and that such a view may not do so in the foreseeable future. More generally, since problem solving methods have not stabilised over time, their extensive reuse is unlikely.

Is there any evidence in the literature to fault this pessimistic no-reuse argument? Two major studies with problem solving methods reuse are the SPARK/ BURN/ FIREFIGHTER (hereafter, SBF) experiment (Marques *et al.* 1992) and the MeKA study (Runkel 1995). These studies are described below.

Experiments with Reusing Problem Solving Methods In the MeKA study, Runkel describes eight applications using *mechanisms for knowledge acquisition*, or MeKA. Each MeKA divided a problem solving method into data structure knowledge and control knowledge. MeKAs contain four modules:

1. An acquire/ module which gathers information such as a formula.
2. A verify module that checks it
3. A generalise module which tries to apply the new knowledge to more general expressions, e.g. is the formula applicable to other parameters?.
4. A dialogue module which handles the screen design for the other modules.

All the MeKAs had to be built for the first application (0 percent reuse), but MeKA reuse in subsequent applications rose as high as 88 percent. The Runkel results are shown in Table 2.

In the SBF toolkit, SPARK builds a domain-specific knowledge acquisition tool that is tailored to the business information supplied by the user. BURN conducts a structured interview with the expert. This interview maps the business information offered by the user into a library of inference sub-routines (called mechanisms). The mapping process is guided by problem solving method meta-knowledge. At choice points in the mapping, SBF can ask the user questions which select different problem solving methods. Once this mapping has been made, a rule base can be generated which solves the business problem. This is given to the FIREFIGHTER environment which assists the user in executing and debugging the operationalised program. Marques et.al. report significantly reduced development times for expert systems using the 13 mechanisms in the SBF toolkit. In the nine applications studied by Marques et.al., development times changed from one to 17 days (using SBF) to 63 to 250 days (without using SBF). To my knowledge, this study represents the high-water mark in reported productivity increases in software or knowledge engineering.

Problems with the Experiments One drawback with the SBF experiment is that it is virtually unrepeatable. Only a few organisations like DEC can spare the personnel to work for nearly a year on throw-away prototypes. The MeKA study is a better experiment than SBF in that the MeKA work has more chance of being reproducible. However, in terms of the Fenton metrics triad, both studies have major drawbacks:

- Poor controls on product measures: There is no success criteria offered for each application. Hence it is possible that the SBF-based application or the MeKA-based systems are somehow inferior.
- Poor controls on resource measures: In human-in-the-loop knowledge acquisition systems, the developers of a system should try the system out on other people. Otherwise, we could encounter the *resource conflation problem*: i.e. the results could confuse the skill of the developer with the intrinsic value of the tool. Who were the personnel who worked on the SBF development or the MeKA tools? If they were the SBF developers and Runkel respectively, then these results suffer from the resource conflation problem. That is, it may be that SBF and MeKAs are only reuse tool for their developers since only they understand their intricacies.

Development order	Application name	$\frac{Reused\ MeKAs}{Total\ MeKAs}$
1	Room assignment	$\frac{0}{7} = 0\%$
2	Elevator configuration	$\frac{5}{8} = 63\%$
3	Elevator design validation	$\frac{7}{8} = 88\%$
4	Configuration validation	$\frac{6}{7} = 86\%$
5	Truck design #1	$\frac{5}{9} = 56\%$
6	Truck pricing	$\frac{10}{12} = 83\%$
7	Truck design #2	$\frac{8}{16} = 50\%$
8	Truck manufacturing	$\frac{15}{17} = 88\%$

Table 2: Reuse in the MeKA system. From (Runkel 1995).

Summary

The evidence for reuse is very weak. After decades of experimentation, transfer has only been observed in the unusual case where some facilitating instructor understands both the new problem and some analogous older problem. Neither the knowledge engineering or software engineering literature adequately demonstrates reuse.

Note that the absence of demonstrable high-levels of reuse does not necessarily prove that change is an overwhelming problem for current knowledge engineering practice. However, given the the poor results to date, despite much widespread effort, it is at least clear that the utility of reuse methods based on expertise transfer is still an open question.

Situated Knowledge Engineering

Our goal is an assessment on the impact of situated cognition on the practice of knowledge engineering. The argument to date has been that mainstream knowledge engineering has yet to demonstrate that change control is a solved problem. This section asks: if we accept managing knowledge base changes as a primary concern, what difference does that make to knowledge engineering? That is, what would be a situated approach to knowledge engineering? In my view, such a situated approach would have at least the following components:

- Modeling the environment around the knowledge base.
- An emphasis on maintenance rather than initial design.
- Metrics to monitor change and the current value of a knowledge base
- The use of *testable* languages.

These components are discussed below.

Model the Environment

In situated knowledge engineering, two knowledge bases are required: the system knowledge base and the environment knowledge base. The system knowledge base is the knowledge base developed in standard practice. The environment knowledge base models the impact of the system knowledge base on its environment (and visa versa). The system and environment knowledge base can interact. Suchman (Suchman 1987; 1993; Agre 1990), for example, argues that real-world planning systems have to model their environment as well as their own goals. For example, a photocopier advisor system must...

> ... focus on the ways in which the photocopier and its user work together to maintain a shared understanding of what is going on between the two of them and the copier... Far from executing a fully operational plan for effecting a fixed goal, the photocopier users continually reinterpreted their situation and based their various actions on their evolving interpretations (Agre 1990).

There are many ways to model the environment around a knowledge base. For example, Clancey et.al.'s BRAHMS system (see this issue) models the exchange of information amongst a group of agents about functional knowledge (orders, organisations, roles, product flows). BRAHMS includes very detailed descriptions of the actual day-to-day work of those agents. Ranesh and Dhar's REMAP system (Ramesh & Dhar 1992) logs design discussions and their inter-connections. If a developer changes their position on some argument, then developers can track the impact of that change to the constraints on the development. Previous discussions can be replayed to generate an historical understanding of how some decision was achieved. User-focused metrics can record how a user will assess the success (or failure) of a knowledge based system. Notes on user-focused metrics are given below.

User-Focused Measures A knowledge base success criteria should reflect end-user concerns and not internal criteria (Buchanan & Shortliffe 1984; Gaschnig *et al.* 1983; Menzies 1998b). Basili (Basili 1992) characterises software evaluation as a goal-question-metric triad. Beginners to experimentation report whatever numbers they can collect without considering the goal of the research project, what questions relate to that goal, and what measurements could be made to address those questions. The generation of user-focused goals is a non-trivial exercise and is rarely reported in the knowledge engineering literature (exception: (Menzies 1998b)).

As an example of user-focused success criteria, the PIGE farm-management expert system (Menzies *et al.* 1992), was not evaluated using internal criteria such as (e.g.) number of productions fired per second. Rather, the success criteria reflected the concerns of the population of farmers who might wish to buy the package: increased profitability per square meter per day.

Often, the success criteria imposes extra requirements on the implementation. We may need to build a very simple initial system that collects baseline measurements which reflect current practice. For example, once I identified increased sales per day as the success criteria for a dealing room expert system. However, this number was not currently being collected in the current software. Sales per day could be estimated from the quarterly statements, but no finer grain data collection was performed at that site. Hence, prior to building the expert system, a database system had to be built to collect the baseline data.

Ripple-Down-Rules, a Maintenance-Focused Approach

Ripple-down-rules (Compton & Jansen 1990; Compton *et al.* 1993) is a maintenance-focused technique that totally de-emphasies initial design. Nevertheless, it has been shown to facilitate easy maintenance in certain domains. Ripples is an excellent example that maintenance is not necessarily facilitated by a greater emphasis on initial design.

A ripple-down knowledge base is organised into a *patch tree*. If a rule is found to be faulty, some patch logic is added on a *unless* link beneath the rule. The patch is itself a rule and so may be patched recursively. Whenever a new patch (rule) is added to a ripple down rule system, the case which prompted the patch is included in the rule. These *cornerstone cases* are used when fixing a ripple down rule system. At runtime, the final conclusion is the conclusion of the last satisfied rule. If that conclusion is faulty, then the fault is localised to the last satisfied rule. Once a fault is localised, an expert can then ask the system for a list of possible patches. The system replies with a *difference list* which is calculated as follows. As the *current case* navigates down the ripple down rule tree, if it finds some satisfied rule, it then checks their *unless* patches. The difference list can be found in the difference between the current case and the cornerstone case of the last satisfied rule.

Ripple-based knowledge bases are developed with minimal initial analysis (Compton *et al.* 1993). Once a set of relevant features are recognised in some domain, ripple patch trees are built by the continual patching of rules whose conditions test for these features. The ripple down rule formalism makes no commitment to tree structures that are optimal: a patch tree can contain repeated tests, redundant knowledge, and its sub-trees can overlap each other semantically. Despite these apparent drawbacks, ripples appears to work very well:

- In practice the ripple down rule trees are only twice as big as an optimum tree (Gaines & Compton 1992) and runtimes have never been an issue.
- Only when thousands of examples are available (in one study, 5000) does machine learning techniques perform as well as manually-built ripple down rule systems. When only hundreds of examples were available, ripples clearly out-performs machine learners (Mansuri, Compton, & Sammut 1991).
- The PEIRS ripples-based system at St. Vincent's Hospital, Sydney, modeled 20 percent of human biochemistry sufficiently well to make diagnoses that are 95 percent accurate (Preston, Edwards, & Compton 1993). PEIRS succeeded in domains where previous attempts, based on much higher-level constructs, never made it out of the prototype stage (Patil, Szolovitis, & Schwartz 1981). Further, while large expert systems are notoriously hard to maintain (de Brug, Bachant, & McDermott 1986), the low-analysis approach of ripples has never encountered maintenance problems. System development blends seamlessly with system maintenance since the only activity that the ripples interface permits is patching faulty rules in the context of the last error. For a 2000-rule RDR system, maintenance was very simple (a total of a few minutes each day).

Knowledge Engineering Metrics

Two kinds of metrics are required for situated knowledge engineering. Firstly, we need to track what portions of a system are changed during its life cycle. The case for situated knowledge engineering disappears if, in systems that allow change, change does not occur.

Secondly, in the situated view, knowledge base change is a constant. An issue with changing a working knowledge base is that the changes may not improve a knowledge base. A situated approach to knowledge engineering would constantly generate metrics that monitor the success of the current version of a knowledge base. The rest of this section offers some brief notes on methods of collecting such success measures: comparative assessment, blinding studies, repeated evaluations. Another method, user-focused metrics, was discussed above. For more notes on evaluation methods see (Fenton 1991; Cohen 1995). For introductory remarks to experimental methods, software measurement, and the evaluation of expert systems, see (Reich 1995; Fenton, Pfleeger, & Glass 1994; Gaschnig *et al.* 1983). For examples of good empirical evaluations, see (Yu *et al.* 1979; Corbridge, Major, & Shadbolt 1995; Menzies 1996; Vicente, Christoffersen, & Pereklita 1995; Sanderson, Verhapge, & Fuld 1989). For examples of very good empirical evaluations, see (Hayes 1997; Yost 1992). For further methodological notes, see (Menzies 1998b; Menzies *et al.* 1997; Menzies 1998a).

Comparative Assessment Statements such as *software technology X lets me do task Y* is hardly an evaluation statement. There may be many software technologies that allow us to implement Y. A better statement is comparative: *software technology X lets me do task Y better than software technology Z.* In order to compare an approach, we need to identify an alternative approach. General principles for comparative empirical evaluation of knowledge engineering methods are discussed in (Menzies 1998a). Such comparative evaluations can take the form of:

- Analysing program vs expert performance; e.g. (Hayes 1997; Menzies *et al.* 1992; Yu *et al.* 1979).
- Analysing expert vs expert performance using different tools (e.g. (Corbridge, Major, & Shadbolt 1995)) or records of their knowledge (e.g. (Shaw 1988));

- Analysing the performance of variants within some program either via an empirical average case analysis (e.g. (Waugh, Menzies, & Goss 1997; Menzies, Cohen, & Waugh 1998)) or a theoretical analysis such as graph theory (e.g. (Menzies & Cohen 1997)) or a worst-case time complexity analysis (e.g. (Tambe & Rosenbloom 1994; Levesque & Brachman 1985)).

Blinding Studies We should not ask experts to evaluate a program merely by watching it run. Often, experts disagree about what is the correct knowledge ((Gaines & Shaw 1989; Shaw 1988; Gaschnig *et al.* 1983; Yu *et al.* 1979)). The *halo effect* prevents a developer from looking at a program and assessing its value. Cohen likens the halo effect to a parent gushing over the achievements of their children and comments that...

What we need is not opinions or impressions, but relatively objective measures of performance (Cohen 1995), p74.

The opposite of the halo effect is when the recommendations of the expert system are rejected merely because some judge knows that the recommendations come from a computer program. (Gaschnig *et al.* 1983) hence recommends *blinding* studies. In such blinding studies, the evaluating agent is not told recommendations come from the expert systems and which come from other sources.

Repeated Evaluations After performing one evaluation, your work does not stop there:

- A good experimenter critically reviews their results (if they don't, someone else will) and look for ways to improve them. For example, the MYCIN evaluation study (Yu *et al.* 1979) took five years and two earlier versions to define adequately. Faults with the prior versions were used to design the next version (Buchanan & Shortliffe 1984).
- Evaluation should be repeated whenever the knowledge changes. In situated knowledge engineering, change is a constant process. Hence, evaluating alternate versions of a theory is is also a constant process.

In practice, seemingly trivial variants in a representation can block our ability to tell if theory 1 is any better than theory 2. For situated knowledge engineering, this is undesirable since such evaluations are a constant feature of the life cycle. An experiment is described below which evaluates the power of a representation language to evaluate a theory.

Summary At least the following techniques are recommended for knowledge engineering metrics: repeated evaluations; user blinding; user-focused success metrics; and comparative evaluations between alternate approaches. These techniques are not standard practice in mainstream knowledge engineering. However, they should be standard practice for situated knowledge engineering.

Testable Languages

This section describes experiments which assess the *testability* of a theory. This section is presented for two reasons:

1. The ability to critique a theory multiple times is an essential part of a situated approach to knowledge engineering (recall the arguments above of Shalin et.al.). Limits to testability are hence limits to situated knowledge engineering.
2. It was mentioned in the introduction that Vera and Simon lamented that accepting situated cognition implies rejecting falsifiable scientific hypotheses. The following experiment is a counter-example to that view.

Knowledge representation languages are usually assessed via their *expressibility* (what can be said in some language) and their *tractability* (what are the upper bounds on inference runtime in that language). Often, there are trade-offs between these criteria. Sometimes, seemingly minor variants in expressibility can have disastrous effects on tractability (Levesque & Brachman 1985).

A situated approach to knowledge engineering should add a third assessment criteria to a knowledge representation language. Situated knowledge engineering requires the continual testing of theories. A good language for situated knowledge engineering supports *testability*: the ability to compare two versions of a theory and declare one to be better than the other. This section reviews trade-offs between testability and expressibility. Just as with tractability and expressibility, it will be shown that seemingly minor variants in expressibility can have disastrous effects on testability (Levesque & Brachman 1985).

Defining Testability Testability is a comparative assessment. Consider a theory *Ti* written in two languages. As we revise our theory, we generate *T1*, *T2*, *T3*.... Given some test procedure, we can assess if some theory *Ti* is better than another theory *Tj*. According to the testability requirement, language one will be considered better than language two if:

- In language one we can check that *Ti* is better than *Tj*;
- But in language two, we cannot.

Variants to a Language Elsewhere, Menzies et.al. (Waugh, Menzies, & Goss 1997; Menzies *et al.* 1997; Menzies, Waugh, & Goss 1998) have explored the testability of four qualitative simulation languages. These four languages (XNODE, INODE, XEDGE, IEDGE) are very small variants of each other. However, only some of these languages let us distinguish between good and bad models (see below).

Consider a model of profitability of fishing in Figure 2. Vertices in this model reflect continuous-valued variables. Edges in this model reflect qualitative influence statements. In the fishing model, there are two such influence statements: direct and inverse. The direct influence between *boatMaintenance* and *netIncome* (denoted by plus signs) means that *netIncome* being *up* or *down* could be explained by *boatMaintenance* being *down* or *up* respectively. The inverse influence between (e.g.) *catchProceeds* and *netIncome* (denoted by minus signs) means that *netIncome* being *up* or *down* could be explained by *catchProceeds* being *down* or *up* respectively.

Variants on a qualitative simulation language can be defined via different *temporal linking policies*. When execut-

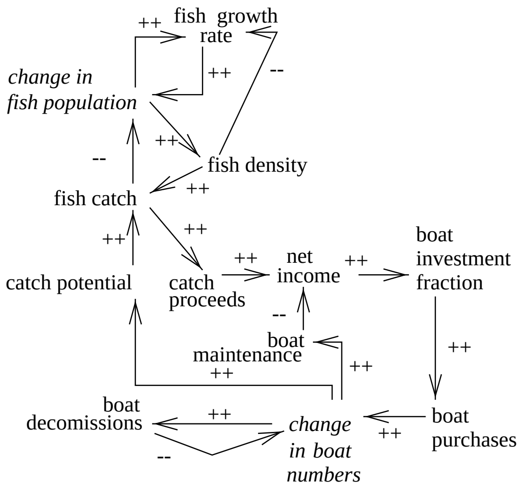

Figure 2: The fisheries model. Adapted from (Bossel 1994) (pp135-141).

ing (e.g.) the fishing model, we need to handle assigning different values to the same variable. For example, consider the feedback loop *direct(change in boatNumbers, boatDecommissions)* and *inverse(boatDecommissions, (change in boatNumbers)*. As a simulation runs, the variables in this feedback loop could take on several values. This can be implemented using a renaming trick. If we run the model for 4 time steps, we can create one copy of each variable for each time step; e.g. *boatDecommissions1... boatDecommissions4, change in boatNumbers1... change in boatNumbers4*.

Once the copied variables are created at different time steps, how are we to connect them? Firstly, it will be assumed that all connections *X to Y* imply a possibly instantaneous connection from *X* to *Y* at the same time; i.e. *X(time=i) to Y(time=i)*. Secondly, we will define four variants on how we connect variables at *time=i* to *time=i+1*. Each way will define a different qualitative simulation language:

- In the implicit edge linking language (or IEDGE), we assume that all edges may take one time tick to traverse. In IEDGE, for all connections *X to Y*, we create a connection *X(time=i) to Y(time=i)*.
- Explicit edge linking (or XEDGE) is a restricted form of IEDGE in which we only create time edges for those edges downstream of *change* variables. The fishing model has two such change variables: *change in boatNumbers* and *change in fishPopulation*. These changes explicitly model the time rate of change of variables. That is, in an explicit time linking policy, time is only traversed near a change variable. So, in XEDGE, for all connections *change in X to Y*, we create a connection *X(time=i) to Y(time=i+1)*.
- In the implicit node linking language (or INODE), it is assumed that some belief now can explain that same belief in the future. That is, for all variables *Z* (either *X* or *change in X*), we create connections *Z(time=i) to Z(time=i+1)*.
- Explicit node linking, or XNODE, is a restricted form of INODE in which only the *change in X* variables have time edges.

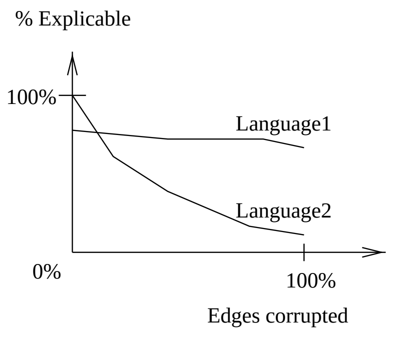

Figure 4: Assessing the relative testability of two languages.

An example of the application of these languages is shown in Figure 3.

Testing Qualitative Theories When running a qualitative model, sometimes we need to make guesses about factors we are unsure off. For example, we may have to guess that *netIncome* in the fishing model goes up, down, or remains steady. Each such guess is mutually exclusive and must be maintained in separate logical worlds. For example, *netIncome=up* would be kept in a separate world to *netIncome=down*. If more than one worlds can be generated, then we need some selection criteria to decide which worlds we prefer; e.g. favour the worlds that contain the largest percentage of the data we wish to explain.

This is a useful test engine for qualitative models. We can now assess a theory via the largest percentage of outputs that can be explained (the *percent explicable* figure). The better the theory, the larger its percent explicable. Feldman and Compton (Feldman, Compton, & Smythe 1989) followed by myself and Compton (Menzies & Compton 1997), have shown that this qualitative test procedure can detect previously unseen errors in theories in neuroendocrinology (the study of nerves and glands) published in international refereed journals. Surprisingly, these faults were found using the data published to support those theories

Assessing Testability The above test procedure can be used to assess testability. Let us corrupt some percentage of the edges in the fishing model (e.g. switch a *direct* to an *inverse* or visa versa). When no edges were corrupted, we had the original fishing model. As more and more edges are corrupted, we can generate worse and worse models. Given the test procedure, we can now perform a comparative assessment of two languages as in Figure 4. The test procedure will supply values for the y-axis of this graph; i.e. what percentage of the outputs can be explained. Values for the x-axis can be supplied by testing models with no corruptions (the original model) to every edge corrupted (a bad model). If a language has a very flat plot on this graph, then it is hard

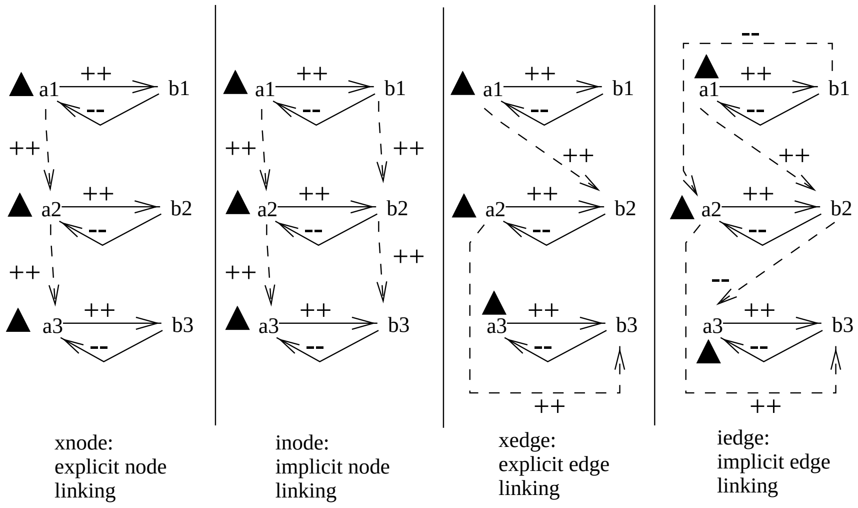

Figure 3: *Direct(change in A, B)* and *inverse(B, change in A)* renamed over 3 time intervals using different simulation languages. The black triangle denotes *change in*. Dashed lines indicate time traversal edges.

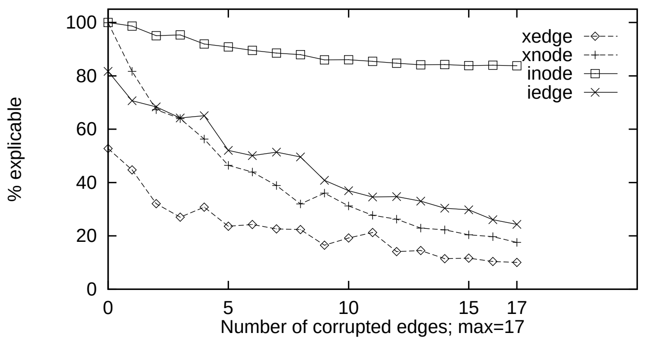

Figure 5: Assessing the relative testability of XNODE, XEDGE, INODE, IEDGE.

to tell good models from bad models. For example, *Language2* will be declared better than *Language1* if the difference between good models and bad models is very clear in *Language2* and very unclear in *Language1*.

Results This testability assessment procedure was applied to XNODE, INODE, XEDGE, IEDGE as follows. The fishing model shown above has 17 influences. Between 0 to 17 of these influences were corrupted (direct to inverse and visa versa). These influences to be corrupted were chosen at random. Hence, the procedure was repeated 20 times for statistically validity. A model for 5 time ticks was then generated for each of INODE, XNODE, IEDGE and XEDGE. This was then executed using 105 data set generated from the quantitative version of the fishing model (Bossel 1994), pp135-141. In all, 18*20*4=1440 models were executed 105 times each; i.e. 151,200 runs.

The results are shown in Figure 5. INODE has a very flat plot; i.e. it is very hard to tell good models from bad models in this language. There is a serious flaw in the XEDGE language: even with no edges corrupted, XEDGE can only explain half the data. XNODE is a much better language. Not only can it explain all the behaviour with the correct models, but the difference between good XNODE and bad XNODE models is quite clear. Note that with less than a third of the fishing influences corrupted (edges corrupted=5), the explicable rate for XNODE fell from 100 to 50 percent. That is, the XNODE language offers a very clear early warning if the model moves away from the correct model. IEDGE was almost as good as XEDGE, however it can only explain 80 percent of the correct data.

Summary This assessment of testability lets us rank the four variants on our simulation language as follows: XNODE (best), then IEDGE (fair), then XEDGE (poor) INODE (poor). More generally, this study has shown that seemingly trivial variants in a language can have disastrous effects on testability. Situated knowledge engineers need to be very aware of the tradeoffs between expressibility and testability.

Conclusion

Situated cognition is not a mere philosophical concern: it has pragmatic implications for current practice in knowledge acquisition.

- A situated knowledge engineer cares less for reusing past knowledge bases than continually adapting existing knowledge bases. Such reuse implies reusing abstracted forms of knowledge extracted from old knowledge bases. This is a synonym for expertise transfer and transfer only has been observed to occurs in the rare case where an instructor can facilitate the transfer.
- A weakly situated knowledge engineer need not abandon the symbolic modeling paradigm. Descriptions of knowledge are very useful for (e.g.) reflecting over what-if scenarios. However, such descriptions are prone to large scale change. A situated knowledge engineer must demonstrate that knowledge base change can be man-

aged by their methodologies while maintaining knowledge base quality. This has two implications:

1. An on-going user-focused metrics programme to continually assess the value of the current version of the knowledge base.
2. Carefully trading off knowledge representation options so as to maximise the testability of a languages.

Finally, Vera and Simon are wrong when they argue that situated cognition is unscientific and un-falsifiable. In the previous section, a repeatable experiment in situated knowledge representation was presented. Also, it is a falsifiable claim that situated cognition impacts on knowledge acquisition. The arguments in this paper would collapse if the following were observed:

- The contents of knowledge bases in working systems do not change significantly over their life time.
- Direct expertise transfer was observed. For example, descriptions of knowledge from old situations can be effectively applied verbatim in new situations.

Acknowledgements.

The comments of Bill Clancey, and Paul Compton helped clarify early versions of this paper. Also, thanks to Chuck Untulis for his careful reading of earlier drafts

References

Agre, P. 1990. Book review: Lcy a. scuhman, plans and situated actions: The problems of human-machine communication. *Artificial Intelligence* 43:369–384.

Anderson, J. 1990. *Cognitive Psychology and its Implications*. W.H. Freeman and Company. 3rd edition.

Bachant, J., and McDermott, J. 1984. R1 Revisited: Four Years in the Trenches. *AI Magazine* 21–32.

Basili, V. R. 1992. The experimental paradigm in software engineering. In Rombach, H. D.; Basili, V. R.; and Selby, R. W., eds., *Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop, Germany*, 3–12.

Benjamins, V., and Jansweijer, W. 1994. Toward a competence theory of diagnosis. *IEEE Expert* 9(5):43–53.

Benjamins, R. 1995. Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research & Applications* 8(2):93–120.

Birnbaum, L. 1991. Rigor Mortis: A Response to Nilsson's 'Logic and Artificial Intelligence'. *Artificial Intelligence* 47:57–77.

Bossel, H. 1994. *Modeling and Simulations*. A.K. Peters Ltd. ISBN 1-56881-033-4.

Bradshaw, J.; Ford, K.; and Adams-Webber, J. 1991. Knowledge representation of knowledge acquisition: A three-schemata approach. In *6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, ,October 6-11 1991, Banff, Canada*, 4.1 – 4.25.

Breuker, J., and de Velde (eds), W. V. 1994. *The CommonKADS Library for Expertise Modelling*. IOS Press, Netherlands.

Brooks, R. 1991. Intelligence without representation. *Artificial Intelligence* 47:139–159.

Buchanan, B., and Shortliffe, E. 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison Wesley. chapter 10. Uncertainty and Evidential Support, 209–232.

Buchanan, B., and Smith, R. 1989. Fundamentals of Expert Systems. In A. Barr, P. C., and Feigenbaum, E., eds., *The Handbook of Artificial Intelligence, Volume 4*, volume 4. Addison-Wesley. 149–192.

Campbell, A.; Hollister, V.; Duda, R.; and Hart, P. 1982. Recognition of a Hidden Material Deposit by and Artificially Intelligent Program. *Science* 217:927–929.

Chandrasekaran, B.; Johnson, T.; and Smith, J. W. 1992. Task structure analysis for knowledge modeling. *Communications of the ACM* 35(9):124–137.

Clancey, W. 1985. Heuristic Classification. *Artificial Intelligence* 27:289–350.

Clancey, W. 1987. Book review of winograd & flores, understanding computers and cognition: A new foundation for design. *Artificial Intelligence* 31:233–250.

Clancey, W. 1991. Book review of israel rosenfield, the invention of memory: A new view of the brain. *Artificial Intelligence* 50:241–284.

Clancey, W. 1992. Model Construction Operators. *Artificial Intelligence* 53:1–115.

Clancey, W. 1993. Situated Action: A Neuropsychological Interpretation (Response to Vera and Simon). *Cognitive Science* 17:87–116.

Cohen, P. 1995. *Empirical Methods for Artificial Intelligence*. MIT Press.

Compton, P., and Jansen, R. 1990. A Philosophical Basis for Knowledge Acquisition. *Knowledge Acquisition* 2:241–257.

Compton, P.; Horn, K.; Quinlan, J.; and Lazarus, L. 1989. Maintaining an expert system. In Quinlan, J., ed., *Applications of Expert Systems*, 366–385. Addison Wesley.

Compton, P.; Kang, B.; Preston, P.; and Mulholland, M. 1993. Knowledge acquisition without analysis. In *European Knowledge Acquisition Workshop.*

Corbridge, C.; Major, N.; and Shadbolt, N. 1995. Models Exposed: An Empirical Study. In *Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge Based Systems.*

de Brug, A. V.; Bachant, J.; and McDermott, J. 1986. The Taming of R1. *IEEE Expert* 33–39.

de Velde, W. V. 1993. Issues in knowledge level modeling. In David, J.-M.; Krivine, J.-P.; and Simmons, R., eds., *Second Generation Expert Systems*. Springer Verlag.

Dreyfus, H. 1979. *What Computers Can't D: A Critique of Artifical Reason*. Freeman.

Duda, R.; Hart, P.; and Reboh, R. 1985. Letter to the editor. *Artificial Intelligence* 26:359–360.

Feigenbaum, E., and McCorduck, P. 1983. *The Fifth Generation*. Addison-Wesley.

Feldman, B.; Compton, P.; and Smythe, G. 1989. Hypothesis Testing: an Appropriate Task for Knowledge-Based Systems. In *4th AAAI-Sponsored Knowledge Acquisition for Knowledge-based Systems Workshop Banff, Canada.*

Fenton, N.; Pfleeger, S.; and Glass, R. 1994. Science and Substance: A Challenge to Software Engineers. *IEEE Software* 86–95.

Fenton, N. E. 1991. *Software Metrics*. Chapman and Hall, London.

Fischer, G.; Lemke, A.; McCall, R.; and Morch, A. 1996. Making argumentation serve design. In Moran, T., and Carroll, J., eds., *Design Rationale: Concepts, Techniques, and Use*. Lawerence Erlbaum Associates. 267–293.

Frakes, W., and Fox, C. 1995. Sixteen questions about software reuse. *Communications of the ACM* 38(6):75–87.

Gaines, B., and Compton, P. 1992. Induction of ripple down rules. In *Proceedings, Australian AI '92*, 349–354. World Scientific.

Gaines, B., and Shaw, M. 1989. Comparing the conceptual systems of experts. In *IJCAI '89*, 633–638.

Gaschnig, J.; Klahr, P.; Pople, H.; Shortliffe, E.; and Terry, A. 1983. Evaluation of expert systems: Issues and case studies. In Hayes-Roth, F.; Waterman, D.; and Lenat, D., eds., *Building Expert Systems*. Addison-Wesley. chapter 8, 241–280.

Gick, M., and Holyoak, K. 1980. Analogic problem solving. *Cognitive Psychology* 12:306–355.

Gruber, T. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5(2):199–220.

Hayes, J., and Simon, H. 1977. Psychological differences among problem isomorphs. In Costellan, N.; Pisoni, D.; and Potts, G., eds., *Cognitive Theory*, volume 2. Hillsdale N.J. Erlbaum. 21–41.

Hayes, C. 1997. A study in solution quality human expert and knolwedge-based system reasoning. In Feltovich, P.; Ford, K.; and Hoffman, R., eds., *Expertise in Context*. MIT PRess. chapter 14, 339–362.

Hoffman, R.; Feltovich, P.; and Ford, K. 1997. A general framework for conceiving of expertise in expert systems in context. In Feltovich, P.; Ford, K.; and Hoffman, R., eds., *Expertise in Context*. MIT PRess. chapter 24, 543–580.

Jacobson, I.; Christerson, M.; Jonsson, P.; and Overgaard, G. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

Lave, J. 1988. *Cognition in Practice*. Cambridge University Press.

Levesque, H., and Brachman, R. 1985. A fundamental tradeoff in knowledge representation and reasoning (revised version). In Brachmann, R., and Levesque, H., eds., *Readings in Knowledge Representation*. Palo Alto: Morgan Kaufmann. 41–70.

Linster, M., and Musen, M. 1992. Use of KADS to Create a Conceptual Model of the ONCOCIN task. *Knowledge Acquisition* 4:55–88.

Mansuri, Y.; Compton, P.; and Sammut, C. 1991. A comparison of a manual knowledge acquisition method and an inductive learning method. In Boose, J.; Debenham, J.; Gaines, B.; and Quinlan, J., eds., *Australian workshop on knowledge acquisition for knowledge based systems, Pokolbin*, 114–132. University of Technology, Sydney.

Marcus, S.; Stout, J.; and McDermott, J. 1987. VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking. *AI Magazine* 41–58.

Marques, D.; Dallemagne, G.; Kliner, G.; McDermott, J.; and Tung, D. 1992. Easy Programming: Empowering People to Build Their own Applications. *IEEE Expert* 16–29.

McDermott, D. 1987. A Critique of Pure Reason. *Computational Intelligence* 3:151–160.

Menzies, T., and Cohen, R. 1997. A graph-theoretic optimisation of temporal abductive validation. In *European Symposium on the Validation and Verification of Knowledge Based Systems, Leuven, Belgium*. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/97eurovav.ps.gz`.

Menzies, T., and Compton, P. 1997. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine* 10:145–175. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96aim.ps.gz`.

Menzies, T.; Black, J.; Fleming, J.; and Dean, M. 1992. An expert system for raising pigs. In *The first Conference on Practical Applications of Prolog*. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/ukapril92.ps.gz`.

Menzies, T.; Waugh, S.; Goss, S.; and Cohen, R. F. 1997. Evaluating a temporal causal ontology. Submitted to FOIS '97. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/97fois`.

Menzies, T.; Cohen, R.; and Waugh, S. 1998. Evaluating conceptual qualitative modeling languages. In *Banff KAW '98 workshop*. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/97evalcon`.

Menzies, T.; Waugh, S.; and Goss, S. 1998. Taming chatter with relevant envisionments. Submitted to ECAI '98. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/98ecai`.

Menzies, T. 1996. On the practicality of abductive validation. In *ECAI '96*. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96abvalid.ps.gz`.

Menzies, T. 1997. OO patterns: Lessons from expert systems. *Software Practice & Experience* 27(12):1457–1478. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/97probspatt.ps.gz`.

Menzies, T. 1998a. Evaluation issues for problem solving methods. Banff KA workshop, 1998. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/97eval`.

Menzies, T. 1998b. Evaluation issues with critical success metrics. In *Banff KA '98 workshop*. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/97evalcsm`.

Motta, E., and Zdrahal, Z. 1996. Parametric design problem solving. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop*.

Newell, A., and Simon, H. 1972. *Human Problem Solving*. Prentice-Hall Englewood Cliffs, N.J.

O'Hara, K., and Shadbolt, N. 1997. Inerpreting generic structures: Expert systems, expertise, and context. In Feltovich, P.; Ford, K.; and Hoffman, R., eds., *Expertise in Context*. MIT PRess. chapter 19, 449–472.

Patel, V., and Ramoni, M. 1997. Cognitive models of directional inference in expert medical reasoning. In Feltovich, P.; Ford, K.; and Hoffman, R., eds., *Expertise in Context*. MIT PRess. chapter 3, 67–99.

Patil, R.; Szolovitis, P.; and Schwartz, W. 1981. Causal Understanding of Patient Illness in Medical Diagnosis. In *IJCAI '81*, 893–899.

Phillips, L. 1984. A theory of requisite decision models. *Acta Psychologica* 56:29–48.

Preston, P.; Edwards, G.; and Compton, P. 1993. A 1600 Rule Expert System Without Knowledge Engineers. In Leibowitz, J., ed., *Second World Congress on Expert Systems*.

Ramesh, B., and Dhar, V. 1992. Supporing systems development by capturing deliberations during requirements engineering. *IEEE Transactions on Software Engineering* 18(6):498–510.

Reed, S.; Ernst, G.; and Banerji, R. 1974. The role of analogy in transfer between similar problem states. *Cognitive Psychology* 6(3):436–450.

Reich, Y. 1995. Measuring the value of knowledge. *International Journal of Human-Computer Studies* 42(1):3–30.

Runkel, J. 1995. Analyzing tasks to build reusable model-based tools. In *Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop Banff, Canada*.

Sanderson, P.; Verhapge, A.; and Fuld, R. 1989. State-space and verbal protocol methods for studying the human operator in process control. *Ergonomics* 32(11):1343–1372.

Searle, J. 1980. Minds, Brain, and Programs. *The Behavioral and Brain Sciences* 3:417–457.

Searle, J. 1982. The myth of the computer. *The New York Review of Books* 3–6. April 29.

Searle, J. 1995. ’the mystery of consciousness’: An exchange. *The New York Review of Books* 83–84.

Shadbolt, N., and O’Hara, K. 1997. Model-based expert systems and the explanations of expertise. In Feltovich, P.; Ford, K.; and Hoffman, R., eds., *Expertise in Context*. MIT PRess. chapter 13, 315–337.

Shalin, V.; Geddes, N.; Bertram, D.; Szczepkowski, M.; and Dubois, D. 1997. Expertise in dynamic, physical task domains. In Feltovich, P.; Ford, K.; and Hoffman, R., eds., *Expertise in Context*. MIT PRess. chapter 9, 195–217.

Shaw, M. 1988. Validation in a knowledge acquisition system with multiple experts. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1259–1266.

Soloway, E.; Bachant, J.; and Jensen, K. 1987. Assessing the maintainability of xcon-in-rime: Coping with the problems of a very large rule-base. In *AAAI ’87*, 824–829.

Stark, M. 1993. Impacts of Object-Oriented Technologies: Seven years of Software Engineering. *J. Systems Software* 23:163–169.

Steels, L. 1994. How Can we Make Further Progress in Knowledge Acquisition? In Mizoguchi, R.; Motoda, H.; Boose, J.; Gaines, B.; and Compton, P., eds., *Proceedings of the Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, JKAW ’94*, 65–71.

Suchman, L. 1987. Book review of winograd & flores, understanding computers and cognition: A new foundation for design. *Artificial Intelligence* 31:227–232.

Suchman, L. 1993. Response to vera and simon’s situated action: A symbolic interpretation. *Cognitive Science* 17:71–75.

Tambe, M., and Rosenbloom, P. 1994. Investigating production system representations for non-combinatorial match. *Artificial Intelligence* 68(1).

Tansley, D., and Hayball, C. 1993. *Knowledge-Based Systems Analysis and Design*. Prentice-Hall.

Vera, A., and Simon, H. 1993a. Situated action: A response to reviewers. *Cognitive Science* 17:77–86.

Vera, A., and Simon, H. 1993b. Situated Action: A Symbolic Interpretation. *Cognitive Science* 17:7–48.

Vera, A., and Simon, H. 1993c. Situated action: Reply to william clancey. *Cognitive Science* 17:117–133.

Vicente, K.; Christoffersen, K.; and Pereklita, A. 1995. Supporting operator problem solving through ecological interface design. *IEEE Transactions of Systems, Man, and Cybernetics* 25(4529-545).

Waugh, S.; Menzies, T.; and Goss, S. 1997. Evaluating a qualitative reasoner. In Sattar, A., ed., *Advanced Topics in Artificial Intelligence: 10th Australian Joint Conference on AI*. Springer-Verlag.

Wielinga, B.; Schreiber, A.; and Breuker, J. 1992. KADS: a Modeling Approach to Knowledge Engineering. *Knowledge Acquisition* 4:1–162.

Winograd, T., and Flores, F. 1987. On understanding computers and cognition: A new foundation for design: A respose to the reviews. *Artificial Intelligence* 31:250–261.

Yost, G. 1992. *TAQL: A Problem Space Tool for Expert System Development*. Ph.D. Dissertation, Computer Science, Carnegie Mellon.

Yu, V.; Fagan, L.; Wraith, S.; Clancey, W.; Scott, A.; Hanigan, J.; Blum, R.; Buchanan, B.; and Cohen, S. 1979. Antimicrobial Selection by a Computer: a Blinded Evaluation by Infectious Disease Experts. *Journal of American Medical Association* 242:1279–1282.