Submitted to the Handbook of Software Engineering and Knowledge Engineering,
Volume 2 (see http://www.ksi.edu/seke/hand.html)
URL= http://tim.menzies.com/pdf/00getknow.pdf.

Knowledge Elicitation: the State of the Art

Tim Menzies,

Department of Electrical & Computer Engineering
University of British Columbia, Canada
<tim@menzies.com>

November 28, 2000

1. Introduction

Knowledge acquisition (KA) is a well-defined process. However, that definition may be too narrow to support an evaluation of the state of the art [66]. The majority of the KA community, including Shadbolt et.al. [103], would define KA as follows:

- KA is usually the process of mapping expert statements into conceptual models.
- KA usually tries to reuse older conceptual models to guide the development of new models.
- KA is usually an early life cycle activity that precedes much of the implementation details.

While exceptions do exist*, the above three points characterize of the majority of the work seen in the Japanese, North American, European, and Pacific region KA workshops†. Despite several major international collaborative projects (the Sisyphus experiments [59, 99, 103]), little has been been concluded about the relative merits of different KA techniques. Shadbolt et.al. comment pessimistically that...

> ... none of the Sisyphus experiments have yielded much evaluation information [103].

Shadbolt's pessimism is unfounded, but only if researchers look beyond standard KA literature. It is true that comparative statements about KA techniques are few and far between in the standard KA literature [66]. However, by looking further afield, this chapter will make several comparative statements about different techniques.

Figure 1 places KA in a broader context. This broader context can be called the *knowledge elicitation* (KE) space. The KE space of Figure 1 has two dimensions. The *how* dimension refers to how the elicitation process is organized. For example, if the developers make extensive use of a library of components, then *how=reuse*. The *what*

* See the work of the "Australian KA club"; e.g. [13, 24, 27–29, 37, 54, 55, 58, 69, 71–73, 75, 77, 79, 94, 95, 115].

† For proceedings of these workshops see http://ksi.cpsc.ucalgary.ca/KAW/.

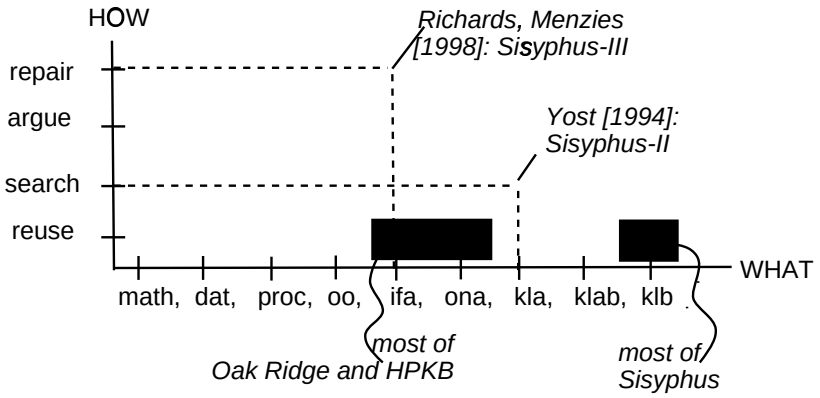

Figure 1: A The *how* and *what* of contemporary KE. Shown on this plot are various studies discussed in this chapter that explore part of this space. For an explanation of the terms used on the x-axis and y-axis, see the text.

dimension refers to the primary modeling construct used by the analysts. For example, if after talking to experts analysts write down object-oriented (OO) class diagrams, then *what=OO*.

The black boxes shown on the KE space of Figure 1 mark areas that have been extensively studied in the literature. The key feature of Figure 1 is the large unexplored regions: most of the evaluation experiments in this area comes from a small subset of the KE space. Objects that are close together are hard to distinguish. It is easier to distinguish differences between objects held far apart. The KE space includes more techniques that found in the standard KA literature. Within this broader space, data points can be found that let us assess contemporary KA and KE techniques.

This chapter will conclude that there are many open issues in contemporary KE that require more investigation and evaluation. Numerous texts describe effective methods for the evaluation of software in general [39] and AI software in particular [23]. For pointers to resources and examples of KE evaluations, see [66]. Evaluating KE techniques is not as hard as many researchers fear. For example, cost-effective methods of studying complex systems are discussed in [65, 73].

2. What is the Form of the Elicited Knowledge?

This exploration of the KE space begins with a definition of the *what* axis of Figure 1. This axis shows what analysts write down when they record knowledge. Numerous notations exist such as the ones shown in Figure 2 and discussed below: MATH, DAT, PROC, OO, IFA, ONA, KLA, KLAB, and KLB. Most of contemporary KE research focuses on OO, IFA, ONA, and KLB.

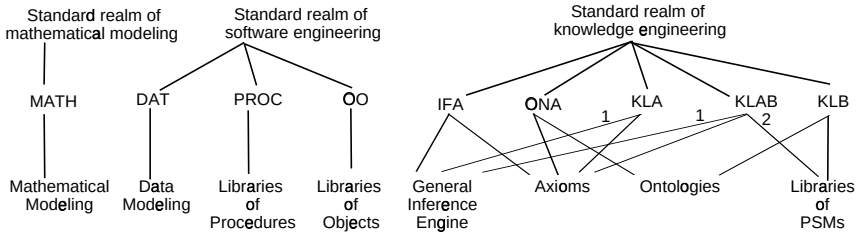

Figure 2: *What* is written down as "knowledge". "1" denotes that the single inference engine is customizable; e.g. the knowledge engineer can provide operator selection rules to customize the problem space traversal [57]. "2" denotes that PSMs in KLAB are used only in an initial analysis stage.

2.1. *What = MATH : mathematical representations*

Impressive and intuitive visual programming environments exist that allow end-users to model complex systems, then execute them using some mathematical equation solver [45, 50]. One advantage of the MATHs-based approach is that certain quality factors (stability, observability, controllability) can be rigorously determined [51].

An interesting variant on standard quantitative MATHematical modeling is qualitative modeling. Qualitative reasoning is the study of equations of physicals systems whose numeric values are replaced by one of three qualitative states: *UP*, *DOWN* or *STEADY* [52]. Qualitative models have at least two advantages over quantitative models and one major disadvantage. The advantages are:

1. Because they contain less details, qualitative modeling can be performed cheaply and early in a product's life cycle [117]. For example it is faster to elicit the qualitative knowledge that (e.g.) `a encourages b` than the quantitative knowledge that $b = 2.4\pi^{1.01*a}$

2. While it can be difficult explaining quantitative equations, simple explanation structures can be easily generated from qualitative equations [45, 105].

The disadvantage of qualitative modeling is the *chatter problem*. Qualitative models contain far less details and far fewer constraints than quantitative models. Hence, when executed, qualitative models can generate an overwhelming number of possible behaviors. Clancy and Kuipers observe that...

> Intractable branching due to irrelevant distinctions is one of the major factors hindering the application of qualitative reasoning techniques to large real-world problems [20].

One method for taming chatter is the DecSIM simulator in which the user divides the theory into several partitions [20]. These partitions are then simulated as separate units. While DecSIM has been able to offer richer simulations than standard qualitative reasoners, DecSIM's authors comment that "DecSIM cannot guarantee a tractable simulation for any model.".

Another method for taming chatter is to impose modeling restrictions that dodge the branching problem. Three such restrictions are *avoiding unreachable models*, a *restricted simulation policy* and the use of *saturation languages*:

- In a *reachable* model, goals can be quickly found before chatter overwhelms the simulation. Detectors for reachable models expressed as directed graphs are discussed by Menzies and Cukic [67]. While these detectors can find chatter-prone models, they offer little help in stopping a model chattering.
- In a *saturation language*, if some property can't be reached after a very limited simulation, it is certain that it cannot be reached after a much longer simulation [76]. That is, any model written in a saturation language is also a *reachable model* (defined above). The disadvantage of these saturating languages (e.g. *iedge* [75, 114]) is that only very simple concepts can be expressed. However, the advantage of such saturation languages is that they can be searched using nearly linear time algorithms [63]. Hence, very large models can be processed. Further, despite their language restrictions, they are adequate to represent a wide range of interesting models such as propositional logical models, early life cycle software requirements [63], and qualitative medical knowledge about internal human physiology.
- A qualitative simulator can execute faster with less chatter if it restricts itself to only those behaviors that lead to pre-specified goals. This *restricted simulation policy* has been used to find previously undetected faults in theories published in international refereed medical journals [37, 68, 78]. Note that such a restricted simulation policy is not required for a model written in a saturation language.

2.2. *What = DAT : representations from data modeling*

Data modeling researchers assume that knowledge will be expressed in something like a relational databases. Theoretically, there is nothing stopping data modeling workers from developing knowledge based systems (KBS) [110]. However, in practice, conventional database manipulation languages are much stronger on IO functions and disc storage than intricate RAM-based manipulation of data. Nevertheless, the relational model offers impressive support for structuring knowledge such that it can be better maintained [33].

2.3. *What = PROC or OO : procedural or object-oriented representations*

A popular method of expressing knowledge is in some procedural (PROC) or object-oriented (OO) form. PROC and OO researchers reject the declarative representations used in the other approaches. In the 70s, this was a large research area. Proponents of *frame* representations (e.g. [81, 118]) argued that part of human expertise was "know-how" and these recipes of "how" to solve a problem were best modeled as (e.g.) Lisp procedures attached to frame slots. The debate continues to this day [7,88] but the complexity of reasoning about procedures (e.g. [36]) drove most researchers to declarative characterizations of their frame-based knowledge (e.g. [9]).

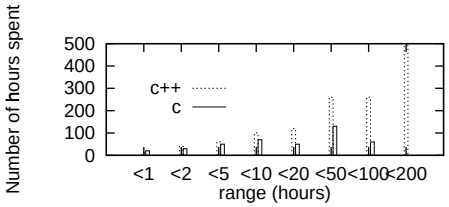

Figure 3: Maintenance times: $OO > procedural$. From [48].

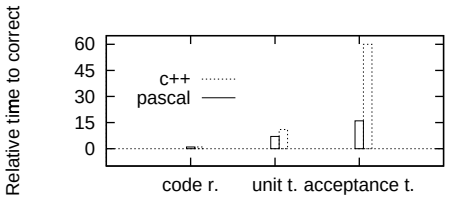

Figure 4: Testing times: $OO > procedural$. "Code r"= time spent on code reviews. "Unit t."= time spent on unit testing. "Acceptance t"= time spent on acceptance testing. From [49].

Pure PROC/OO KE researchers are rare these days, but some still keep the faith (e.g. [7, 10]). OO has become a very popular modeling paradigm in the commercial world [8]. OO languages are useful for many task; e.g. JAVA is a natural language for concurrent web-based applications. However, OO class hierarchies can *confuse* rather than *clarify* software maintenance. Hatton [48] reports one study over a six year period with a ≈50,000 LOC parser written and maintained in C++ and C. The programmers on this team were experienced in both language and used mature software processes (CMM-level 2^+). The average time chasing bugs in the C++ parser was much greater than in the C version (see Figure 3). Hatton acknowledges that his results cannot distinguish between the effects of object-oriented hierarchies and the particulars of the implementation environments. However, he notes that analogous results have been reported elsewhere [49]; i.e. a C++ system was much slower to change later in the life cycle than an equivalent procedural system (see Figure 4).

Hatton's explanation for these empirical results casts doubt on the utility of inheritance as a modeling construct. His argument is that the increased maintenance times of C++ are not due to the idiosyncrasies of the language. Rather, these problems arise from the distributed nature of properties in an inheritance hierarchy. Debugging such a hierarchy requires tracing up and down a hierarchy. During that trace, to understand some method in class X, the programmer must also *keep in mind* what is known about how class X effects a method in class Y. Hatton's argument is this *keeping in mind* process is slow and difficult for people since it requires reading and writing to long-term memory (something humans do very slowly and very poorly).

2.4. *What = IFA : inference over axioms*

In the IFA approach, there are domain facts and, usually, no explicit generalization of those facts. IFA typically commits to a single inference procedure. Examples of

these inference engines include Prolog [56], OPS5 [42], SOAR [57, 97], PSCM [120], GSAT [101], ISAMP [32],....

Crudely expressed, in the IFA approach, knowledge engineering is just a matter of stuffing axioms into an inference engine and letting the inference engine work it all out. IFA was how knowledge engineering was performed in the 70s. Drawbacks with IFA motivated the development of the other techniques described in this chapter. The critiques of [15] and [17] were particularly influential. These writers argued convincingly that above the level of mere rules, there existed domain-independent organization principles for expert systems. These higher-level constructs are the focus of much of contemporary KE research (see the discussion below on ONA, KLA, KLAB, KLB).

2.5. *What = ONA : use of ontologies*

ONA is short for "ONtologies + Axioms". Axioms are some logical assertion describing facts or rules in a domain. Ontologies model common domain terminology. This terminology might include the data structures required by a problem solving method (PSM- discussed below). Using a good ontology, it is argued, can guide developers in the construction of new systems.

An active focus in ONA is the creation of ontologies. Ontologies may never execute- rather they may be an analysis tool for a domain. Software engineers who develop architectures or design patterns, but do not execute these abstractions directly, are ONA (e.g. [44]).

Ontologies are a large area of active research. For more details on this work, see [53].

2.6. *What = KLA : Newell's knowledge-level*

KLA modeling adopts Newell's *knowledge-level* insight [86, 87]. According to Newell, intelligence may be modeled at the knowledge level as a search for appropriate operators that convert some current state to a goal state. Domain-specific knowledge is used to select the operators according to *the principle of rationality*; i.e. an intelligent agent will select an operator which its knowledge tells it will lead the achievement of some of its goals.

KLA makes a strong commitment to a single inference procedure, which can be customized. This inference procedure features predominantly when modeling a system; e.g. [57, 80, 120]. For example, a KLA analyst would record the different states a system can take, the choices at each state, and the rationality operators that select how to move from this state towards the goal state.

2.7. *What = KLB : other knowledge-level modeling approaches*

KLB modeling assumes that new applications are built by reusing old problem solving methods (PSMs) and ontologies. A sample PSM is shown in Figure 5. KLB modeling proceeds by first selecting a PSM, then collecting domain specific information relating to the data types in the selected PSM. For example, using this PSM, an analyst would be guided to ask "what are the `norm` values for the `observables`?".

KLB uses either libraries of PSMs [116] or explores a single PSM within such a

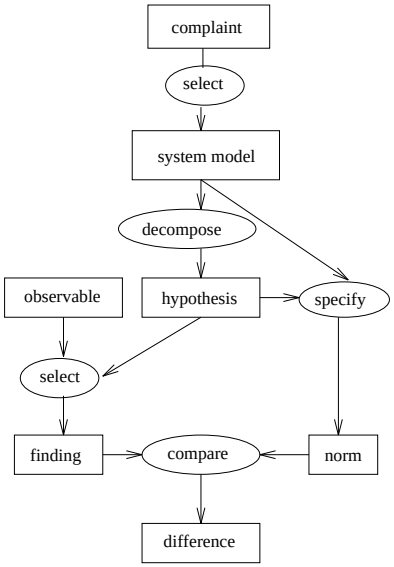

Figure 5: A problem solving method (PSM) for diagnosis; ovals=functions, rectangles=data types.

library [60]. KLB makes extensive use of ontologies. At runtime, KLB may use a general inference engine to execute their systems (e.g. older versions of PROTEGE-II [35] compiled down to CLIPS [85]) but this inference engine does not feature in the design discussions.

Numerous libraries of PSM have been collected; e.g SPARK/ BURN/ FIREFIGHTER [61]; generic tasks [16]; configurable role-limiting methods [46, 108]; model construction operators [19]; CommonKADS [100, 116]; the PROTEGE family of systems [35]; components of expertise [106]; MIKE [4]; and TINA [6].

There is a large research community working on KLB and this chapter is too short to discuss all that work. Another chapter in this handbook is especially devoted to KLB [83]. Note that the KLB community calls its activities "knowledge-level modeling" even though significant differences exist between KLB-style knowledge level modeling and Newell-style KLA knowledge level modeling:

- Newell-style knowledge-level modeling is based on customizing a single search engine.
- The PSM/ontology community is based on the reuse of libraries of PSMs and ontologies.

2.8. *What = KLAB*

KLAB is a hybrid KLA/KLB approach in which PSMs are used to structure the analysis discussions, but then converted by the knowledge engineer at design time to KLA [16].

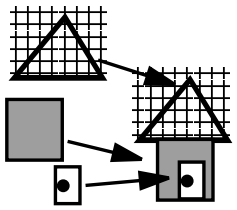

Figure 6: An example of *what=reuse*. A house is built using parts found when the last house was built.

3. How is Knowledge Elicited?

Having described the *what* of the KE space, it is now time to study the vertical *how* axis of Figure 1. Moran and Carroll [82, p3] argue that four broad paradigms of exist in the literature which we will call *reuse, search, argue, repair*. These paradigms are described below.

3.1. *How = Reuse*

How=reuse dates back at least to 1964 with Alexander's work on architecture [2, 3]. Design is taken to be the re-shuffling of components developed previously, then abstracted into a reusable form. For example, in Figure 6, a new house is built by reflecting over models developed when older houses were built.

Contemporary expressions of this approach include object-oriented design patterns [11, 70], and the knowledge engineering research into ontologies [47, 113] and problem solving methods (PSMs) [15, 17, 100]. Our reading of the literature is that this approach is the dominant paradigm in contemporary knowledge and software engineering.

KLB and ONT assume *how=reuse*. The fundamental premise of the KLB and ONT research is that conceptual models abstracted from old domains are power tools when reused in new domains.

3.2. *How = Search*

The paradigm of construction-as-search dates back at least to 1969 with Simon's work on artificial intelligence [104]. According to Simon, design is the traversal of a space of possibilities, looking for pathways to goals. For example, to build a house, we might search through a space of constraints describing the relationship between building parts (see Figure 7).

Contemporary expressions of this approach includes most of the AI search literature [90]. KLA assumes *how=search*, often using the SOAR toolkit [86, 87] with or without the PSCM extension [120].

3.3. *How = Argue*

This paradigm dates back to at least 1972 with Ritell's work on *wicked prob-*

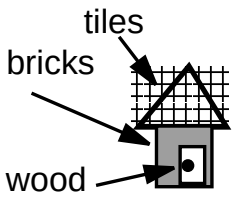

Figure 7: An example of *how=search*. A house is built via a search of the space of what is known of bricks, tiles, and wood.

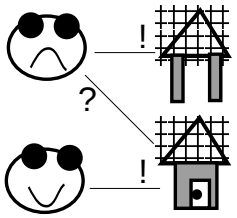

Figure 8: An example of *how=argue*. Let each stakeholder design their preferred house separately. The merits of the different houses can then then be assessed.

lems [96]. Wicked problems have many features, the most important being that no objective measure of success exists. Designing solutions for wicked problems cannot aim to produce some perfectly correct answer since no such definition of *correct* exists. Hence, this approach to design tries to support effective debates by a community over a range of possible answers. For example, when building a house using *how=argue*, different stakeholders first elaborate their own version of the final product. The different versions are then explored and assessed (see Figure 8).

Contemporary expressions of this approach includes the requirements engineering (RE) community. Requirements engineering is usually complicated by the incompleteness of the specification being developed: while a specification should be consistent, requirements are often very inconsistent. RE researchers such as [34], [40], and [89] argue that we should routinely expect specifications to reflect different and inconsistent viewpoints. Note that this paradigm requires more than one expert to be part of the knowledge acquisition process.

3.4. *How = Repair*

This paradigm dates back to at least 1983 with Schon's work on the *reflective practitioner* [98]. In this approach, design mostly happens when some concrete artifact *talks back* to the designer- typically by failing in some important situation. That is, *how=repair* is less concerned with the creation of some initial artifact than the on-

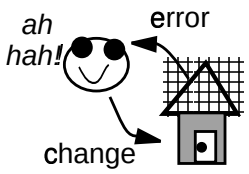

Figure 9: An example of *how=repair*. Build a house, find out that the living room gets too hot in summer, plant fruit trees for shade, make and sell the jam from the fruit.

going re-interpretation and adjustment of that artifact. So, when building a house using *how=repair*, version i+1 of the house is created using the lessons learnt from building version i.

Contemporary expressions of this approach include the situated cognition community [18, 74, 107], certain approaches to design rationale [12, 41], and knowledge engineering techniques that focus on maintenance rather than initial design. For example, in the *ripple-down-rules* (RDR) repair-based system, knowledge is organized into a *patch tree*. If a rule is found to be faulty, some patch logic is added on an *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 an RDR system, the case which prompted the patch is included in the rule. These *cornerstone cases* are used below when fixing an RDR system. At runtime, the final conclusion is the conclusion of the last satisfied rule. If that conclusion is faulty, then the fault is localized to the last satisfied rule. For more details on ripple-down-rules, see [25, 26, 30, 43, 84, 92, 93].

4. Evaluations

The above discussion outlined what KE researchers have been *doing*. However, Cohen warns:

> Researchers are very good at describing what they are *doing*, but not what they are *learning*. [23]

Hence, we need now to review what we have *learnt* after all that *doing*. The sequel looks for systems that use different combinations *how*'s and *what*'s. Some of those systems come from the Sisyphus work. By comparing these systems, we can at least identify the open issues in contemporary KE.

4.1. *Need More Than "Just" Rules?*

In the section *what=IFA* it was commented that numerous researchers developed KLA, KLB and ONT approaches based on difficulties with IFA. It is an open issue whether the methods developed since IFA are more productive than old-fashioned IFA:

- In the Oak Ridge study, various researchers tried to build applications for the same problem. Most of the groups at Oak Ridge used IFA-based techniques

and produced working prototypes. However, the group using an OO-based tool (RLL) did not produce a working system since most of their time was spent configuring the extensive representational options within their tool [5].

- In the Sisyphus-III initiative [103], Richards and Menzies [95] used an IFA RDR approach to produce a working system for lunar rock classification (RDR was introduced above in the section *what=repair*). In the same time frame, various reuse-based teams produced no working system since they were still debugging their upper-level ontologies.

The comparatively better performance of RDR for lunar rock classification in Sisyphus-III suggests the possibility that *how=repair* may be more productive than than *how=reuse* when building a system. Another data point supporting that possibility comes from DARPA's High Performance Knowledge Base initiative (HPKB) [14, 21, 22, 91]:

- The goal of HPKB was to increase the rate at which a KE tool can collect axioms. Various teams participated in HPKB including many that used *how=reuse* methods. In HPKB year one, the George Mason team generated the most new axioms added per day (787 binary predicates) using DISCIPLE: an incremental knowledge acquisition tool [109]. DISCIPLE includes elaborate repair tools. Inside DISCIPLE are machine learning tools that abstract the supplied knowledge. The abstracted rules are then proposed to the user in order to generate rules which makes them more generally applicable. DISCIPLE requires meta-knowledge to execute, but the system builds and updates that meta-knowledge as it runs.

4.2. *Need More Than "Just" Search?*

The distinction between KLA/KLB is the amount of effort devoted to *pre-modeling* the search procedure:

- Reuse of large libraries of problem solving methods or ontologies is not stressed in KLA. In KLA, there is a single hard-wired search procedure with customizable operators.

- In KLB, elaborate libraries of problem solving methods are used. These libraries contain declarative descriptions of procedures used to perform common tasks such as diagnosis (e.g. Figure 5). These libraries are constantly being improved in order to simplify and improve the construction of new applications.

The Sisyphus-II initiative gives us one data point with which to compare KLA and KLB [99]:

- Yost [119] used a KLA tool (PSCMs) to solve an elevator configuration problem. The other participants in the Sisyphus-II study made extensive use of KLB [99]. Yost's reported development times were much less than any of the other teams. It is possible that Yost's extra experience with the domain gave him an advantage (Yost had analyzed the domain extensively prior to system construction). On the other hand, it is also possible that search-based approaches are more productive than reuse-based approaches.

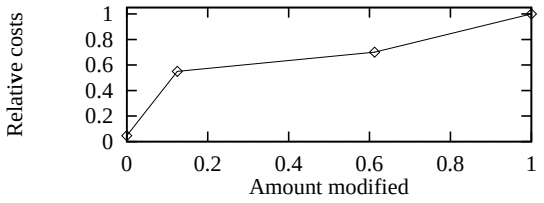

Figure 10: COCOMO-II, the cost of reuse with X% changes . Y-axis shows the relative cost of changing vs rebuilding from scratch. Data from thousands of NASA software modules [1, p21].

The more elaborate the pre-modeling, the harder it is for a newcomer to use and understand the library of reusable components. This effect has been well studied in the SE literature:

- The COCOMO-II software cost estimation model offers an estimate of the cost of adapting reusable sub-routines for a new project [1, p21]. A learning curve must be traversed before a module can be adapted. By the time you know enough to change a little of that module, you may as well have re-written 60% of it from scratch; see Figure 10.

A similar effect may well exist when reusing knowledge-based components:

- Cohen et.al. [21] studied how much ontologies supported the development of HPKB applications. The specific and recent terms added to an ontology offer more support than generic terms added previously by other authors. The former scored a constant 60% rate of reuse in contrast with the poor 22% rate of reuse scored by the broad ontologies. These results cast some doubt on the verbatim reuse benefits of generic ontologies but Cohen et.al. are quick to add that the generic terms may help designers to better structure their systems. However, they offer no empirical evidence to support this possibility.

- In another study, international KA experts used some background knowledge to guide their analysis of a transcript of a patient talking to a doctor [31]. One group could use a supposedly reusable model of diagnosis which had been matured over many years by many researchers. Other groups used either a reuse model of diagnosis invented very quickly (the “straw man”) or no reuse model at all. The results are shown in Figure 11. The “mature model” group performed as well as the “straw man” group. Further, the “no model” group out-performed the groups using the models!

4.3. *ISA is a “Good” Thing?*

The ontologies used in ONT and KLB are often expressed in a type hierarchy. Recalling Figure 3, if such a representation is to be read by a human, it will be fundamentally difficult for that human to find and fix flaws.

Reuse Model	% disorders identified	% knowledge fragments identified
Straw man: invented very quickly	50	28
Mature model: decades of work	55	34
No model	75	41

Figure 11: Productivity using different models.

4.4. *High-Level Constructs are More Testable?*

There are many claims that the high-level constructs of KLB's are easier to test e.g [38, 102, 111, 112]. However, none of this work has yet to devise an experimental evaluation of their claims that *KLB = better testing*. Note that if any of these higher-level constructs are expressed as inheritance hierarchies, then recalling the last section, these hierarchies will be *harder* to manually debug than flatter representations.

While humans *may* be able to read and critique small models, automatic support is required to test models that grow beyond a trivial size. One method for such automatic testing is to search for pathways from known inputs to desired goals or faults. If any faults can be reached or if some goals can't be reached, the system can be faulted. In the case of nondeterministic AI systems, this search could get lost in all the nondeterministic options.

Menzies and Compton [68, 78] have shown that this nondeterministic testing problem can be mapped to the chatter problem. That is, the methods discussed above that solve the chatter problem can also address the problem of automatically testing nondeterministic systems. Recall that all these solutions to the chatter problem discussed above (in the section *what=MATH*) required restrictions on the types of models built, or the language used for the models, or how a model is exercised. Such restrictions are not acceptable to the KLB community. One of the premises of that community is that intelligence can be analyzed using knowledge *content* rather than knowledge *form*; i.e. it is irrelevant if it is expressed in rules or frames or *C* code or inheritance hierarchies or directed graphs or saturation languages. While form *may* be irrelevant for initial knowledge acquisition, it is *vital* when assessing the testability of an artifact.

5. Discussion

This chapter has briefly surveyed a range of techniques used in contemporary KE (for more details, see [62]). One limitation with this survey is that it dealt only with *mostly manual* knowledge elicitation methods that have a *large human-in-the-loop component*. Methods for mostly automatic knowledge elicitation are discussed elsewhere [62–64].

The survey can be used in one of two ways. Firstly, it could be used as a simple road map describing contemporary methods. Secondly, the case studies in the last section might motivate the reader to explore the open issues in KE. Clearly, the case studies are too few to make general conclusions about the utility of different KE techniques. However, the case studies at the very least suggest that we should be more active in critically assessing different KE options.

Acknowledgements

Helen Burgess provided invaluable editorial advice for this article.

References

1. C. Abts, B. Clark, S. Devnani-Chulani, E. Horowitz, R. Madachy, D. Reifer, R. Selby, and B. Steece. COCOMO II model definition manual. Technical report, Center for Software Engineering, USC,, 1998. http://sunset.usc.edu/COCOMOII/cocomox.html#downloads.
2. C. Alexander. *Notes on Synthesis of Form*. Harvard University Press, 1964.
3. C. Alexander, S. Ishikawa, S. Silverstein, I. Jacobsen, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, 1977.
4. J. Angele, D. Fensel, and R. Studer. Domain and task modelling in mike. In A. Sutcliffe et.al., editor, *Domain Knowledge for Interactive System Design*. Chapman & Hall, 1996.
5. D.R. Barstow, N. Aiello, R.O. Duda, L.D. Erman, C.L. Forgy, D. Gorlin, R.D. Greiner, D.B. Lenat, P.E. London, J. McDermott, H. Penny Nii, P. Politakis, R. Reboh, S. Rosenchein, A.C. Scott, W. van Melle, and S.M. Weiss. Languages and tools for knowledge engineering. In F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, editors, *Building Expert Systems*, chapter 9, pages 283–345. Addison-Wesley, 1983.
6. R. Benjamins. Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research & Applications*, 8(2):93–120, 1995.
7. L. Birnbaum. Rigor Mortis: A Response to Nilsson's 'Logic and Artificial Intelligence'. *Artificial Intelligence*, 47:57–77, 1991.
8. G. Booch, I. Jacobsen, and J. Rumbaugh. *Version 1.0 of the Unified Modeling Language*. Rational, 1997. http://www.rational.com/ot/uml/1.0/index.html.
9. R.J. Brachman, V.P. Gilbert, and H.J. Levesque. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton. In J. Mylopoulos and M.L. Brodie, editors, *Readings in Artificial Intelligence and Databases*, pages 293–300. Morgan Kaufmann, 1989.
10. R.A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
11. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley & Sons, 1996.
12. G. Casady. Rationale in practice: templates for capturing and applying design expertise. In T.P. Moran and J.M. Carroll, editors, *Design Rationale: Concepts, Techniques, and Use*, pages 351–372. Lawerence Erlbaum Associates, 1996.
13. J. Catlett. Inductive learning from subsets or disposal of excess training data considered harmful. In *Australian Workshop on Knowledge Acqusition for Knowledge-Based Systems, Pokolbin*, pages 53–67, 1991.
14. M. Ceruti, C. Anken, A. Lin, and S. Rubin. Applications of high-performance knowledge-based technology. In *Proceedings of IEEE Systems Man and Cybernetics*, 2000. Available from http://reliant.teknowledge.com/HPKB/Publications/ceruti.ps.
15. B. Chandrasekaran. Towards a Taxonomy of Problem Solving Types. *AI Magazine*, pages 9–17, Winter/Spring 1983.

16. B. Chandrasekaran, T.R. Johnson, and J. W. Smith. Task structure analysis for knowledge modeling. *Communications of the ACM*, 35(9):124–137, 1992.
17. W. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.
18. W. Clancey. The knowledge level reinterpreted: Modeling how systems interact. *Machine Learning*, 4(3/4):285–293, 1989.
19. W.J. Clancey. Model Construction Operators. *Artificial Intelligence*, 53:1–115, 1992.
20. D.J. Clancy and B.K. Kuipers. Model decomposition and simulation: A component based qualitative simulation algorithm. In *AAAI-97*, 1997.
21. P. Cohen, V. Chaudhri, A. Pease, and R. Schrag. Does prior knowledge facilitate the development of knowledge-based systems? In *AAAI'99*, 1999.
22. Paul Cohen, Robert Schrag, Eric Jones, Adam Pease, Albert Lin, Barbara Starr, David Gunning, and Murray Burke. The darpa high-performance knowledge bases project. *AI Magazine*, 19(4):25–49, Winter 1998.
23. P.R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.
24. P. Compton, G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan, and C. Sammut. Ripple down rules: possibilities and limitations. In *6th Banff AAAI Knowledge Acquisition for Knowledge Based Systems*, 1991.
25. P. Compton, G. Edwards, A. Srinivasan, P. Malor, P. Preston, B. Kang, and L. Lazarus. Ripple-down-rules: Turning knowledge acquisition into knowledge maintenance. *Artificial Intelligence in Medicine*, 4:47–59, 1992.
26. P. Compton, K. Horn, J.R. Quinlan, and L. Lazarus. Maintaining an expert system. In J.R. Quinlan, editor, *Applications of Expert Systems*, pages 366–385. Addison Wesley, 1989.
27. P. Compton, B. Kang, P. Preston, and M. Mulholland. Knowledge acquisition without analysis. In *European Knowledge Acquisition Workshop*, 1993.
28. P. Compton, P. Preston, and B. Kang. The use of simulated experts in evaluating knowledge acquisition, 1995.
29. P. Compton, Z. Ramadan, P. Preston, T. Le-Gia, V.Chellen, M. Mulholland, D.B. Hibbert, P.R. Haddad, and B. Kang. A trade-off between domain knowledge and problem-solving method power,. In *Banff Workshop on Knowledge Acquisition*, 1998. Available from `http://ksi.cpsc.ucalgary.ca/KAW/KAW98/compton`.
30. P.J. Compton and R. Jansen. A Philosophical Basis for Knowledge Acquisition. *Knowledge Acquisition*, 2:241–257, 1990.
31. C. Corbridge, N.P. Major, and N.R. Shadbolt. Models Exposed: An Empirical Study. In *Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge Based Systems*, 1995.
32. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.
33. J. Debenham. *Knowledge Engineering: Unifying Knowledge Base and Database Design*. Springer-Verlag, 1998.
34. S. Easterbrook. *Elicitation of Requirements from Multiple Perspectives*. PhD thesis, Imperial College of Science Technology and Medicine, University of London, 1991. Available from `http://research.ivv.nasa.gov/~steve/papers/index.html`.
35. H. Eriksson, Y. Shahar, S. W. Tu, A. R. Puerta, and M. A. Musen. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326, 1995.
36. D.W. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In

AAAI-83, pages 104–108, 1983.
37. B. Feldman, P. Compton, and G. Smythe. Hypothesis Testing: an Appropriate Task for Knowledge-Based Systems. In *4th AAAI-Sponsored Knowledge Acquisition for Knowledge-based Systems Workshop Banff, Canada*, 1989.
38. D. Fensel and A. Schoenegge. Hunting for assumptions as developing method for problem-solving methods. In *Workshop on Problem-Solving Methods for Knowledge-based Systems, IJCAI '97, August 23.*, 1997.
39. N E Fenton. *Software Metrics*. Chapman and Hall, London, 1991.
40. A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specification. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.
41. G. Fischer, A.C. Lemke, R. McCall, and A.I. Morch. Making argumentation serve design. In T.P. Moran and J.M. Carroll, editors, *Design Rationale: Concepts, Techniques, and Use*, pages 267–293. Lawerence Erlbaum Associates, 1996.
42. C.L. Forgy. RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, pages 17–37, 19 1982.
43. B.R. Gaines and P. Compton. Induction of ripple down rules. In *Proceedings, Australian AI '92*, pages 349–354. World Scientific, 1992.
44. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
45. Patrice O. Gautier and Thomas R. Gruber. Generating explanations of device behavior using compositional modeling and causal ordering. In *AAAI-93*, pages 264–270, 1993.
46. Y. Gil and E. Melz. Explicit representations of problem-soving strategies to support knowledge acquisition. In *Proceedings AAAI' 96*, 1996.
47. T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
48. L. Hatton. Does oo sync with how we think? *IEEE Softwre*, pages 46–54, May/June 1998.
49. W. Humphrey. *A Discipline for Software Engineering*. McGraw Hill, 1995.
50. High Performance Software Inc. ithink 3.0.5, 1994.
51. Y. Ishida. Using global properties for qualitative reasoning: A qualitative system theory. In *Proceedings of IJCAI '89*, pages 1174–1179., 1989.
52. Y. Iwasaki. Qualitative physics. In P.R. Cohen A. Barr and E.A. Feigenbaum, editors, *The Handbook of Artificial Intelligence*, volume 4, pages 323–413. Addison Wesley, 1989.
53. Y. Kalgoglou. Ontologies in software design. In S.K. Chung, editor, *Handbook of Software and Knowledge Engineering (volume 1)*, 2001.
54. B.H. Kang and P. Compton. A maintenance approach to case based reasoning. In M.Keane, J.P. Haton, and M. Manago, editors, *Advances in Case-Based Reasoning, Selected Papers from Second European Workshop, EWCBR-94*, pages 226–239. Springer, Lecture Notes in Artificial Intelligence 984, 1995.
55. B.H. Kang, P. Compton, and P. Preston. Multiple classification ripple down rules: Evaluation and possibilities. In *Proceedings 9th Banff Workshop on Knowledge Acquisition*, 1995.
56. R. A. Kowalski. The early years of logic programming. *Communications of the ACM*, 31(1):38–43, January 1988.
57. J.E. Laird and A. Newell. A universal weak method: Summary of results. In *IJCAI '83*, pages 771–773, 1983.
58. M. Lee, P. Compton, and B. Jansen. Modelling with context-dependant causality.

In *Proceedings of the Second Japan Knowledge Acquisition for Knowledge-Based Systems Workshop, Kobe, Japan*, pages 357–370, 1992.

59. M. Linster and M. Musen. Use of KADS to Create a Conceptual Model of the ONCOCIN task. *Knowledge Acquisition*, 4:55–88, 1 1992.
60. S. Marcus and J. McDermott. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39:1–37, 1 1989.
61. D. Marques, G. Dallemagne, G. Kliner, J. McDermott, and D. Tung. Easy Programming: Empowering People to Build Their own Applications. *IEEE Expert*, pages 16–29, June 1992.
62. T. Menzies. Knowledge maintenance: The state of the art. *The Knowledge Engineering Review*, 14(1):1–46, 1999. Available from `http://tim.menzies.com/pdf/97kmall.pdf`.
63. T. Menzies. Practical machine learning for software engineering and knowledge engineering. In *Handbook of Software Engineering and Knowledge Engineering*, 2001. Available from `http://tim.menzies.com/pdf/00ml.pdf`.
64. T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00ase.pdf`.
65. T. Menzies, E. Sinsel, and T. Kurtz. Learning to reduce risks with cocomo-ii. In *Workshop on Intelligent Software Engineering, an ICSE 2000 workshop, and NASA/WVU Software Research Lab, Fairmont, WV, Tech report # NASA-IVV-99-027, 1999*, 2000. Available from `http://tim.menzies.com/pdf/00wise.pdf`.
66. T. Menzies and F. van Harmelen. Editorial: Evaluating knowledge engineering techniques. *International Journal of Human-Computer Studies, special issue on evaluation of Knowledge Engineering Techniques*, 51(4):717–727, October 1999. Available from `http://tim.menzies.com/pdf/99ekeed.pdf`.
67. Tim Menzies, Bojan Cukic, Harhsinder Singh, and John Powell. Testing nondeterminate systems. In *ISSRE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00issre.pdf`.
68. T.J. Menzies. *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales, 1995. Available from `http://tim.menzies.com/pdf/95thesis.pdf`.
69. T.J. Menzies. Assessing responses to situated congition. In *Proceedings of the 10th Knowledge Acquisition Workshop for Knowledge-Based Systems, Banff,Canada*, 1996. Available from `http://tim.menzies.com/pdf/96sitcog.pdf`.
70. T.J. Menzies. OO patterns: Lessons from expert systems. *Software Practice & Experience*, 27(12):1457–1478, December 1997. Available from `http://tim.menzies.com/pdf/97patern.pdf`.
71. T.J. Menzies. Evaluation issues for problem solving methods. In *Banff Knowledge Acquisition workshop, 1998*, 1998. Available from `http://tim.menzies.com/pdf/97eval.pdf`.
72. T.J. Menzies. Evaluation issues for problem visual programming languages, 1998. Banff KA workshop, 1998. Available from `http://tim.menzies.com/pdf/97evalvp.pdf`.
73. T.J. Menzies. Evaluation issues with critical success metrics. In *Banff KA '98 workshop.*, 1998. Available from `http://tim.menzies.com/pdf/97langevl.pdf`.
74. T.J. Menzies. Towards situated knowledge acquisition. *International Journal of Human-Computer Studies*, 49:867–893, 1998. Available from `http://tim.`

menzies.com/pdf/98ijhcs.pdf.
75. T.J. Menzies, R.F. Cohen, and S. Waugh. Evaluating conceptual qualitative modeling languages. In *Banff KAW '98 workshop.*, 1998. Available from http://tim.menzies.com/pdf/97modlan.pdf.
76. T.J. Menzies, R.F. Cohen, S. Waugh, and S. Goss. Applications of abduction: Testing very long qualitative simulations. *IEEE Transactions of Data and Knowledge Engineering (to appear)*, 2001. Available from http://tim.menzies.com/pdf/97iedge.pdf.
77. T.J. Menzies and P. Compton. Knowledge acquisition for performance systems; or: When can "tests" replace "tasks"? In *Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada*, 1994. http://tim.menzies.com/pdf/banff94.pdf.
78. T.J. Menzies and P. Compton. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine*, 10:145–175, 1997. Available from http://tim.menzies.com/pdf/96aim.pdf.
79. T.J. Menzies and S. Goss. Vague models and their implications for the kbs design cycle. In *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop and Monash University Department of Software Development Technical Report TR96-15*, 1996. Available from http://tim.menzies.com/pdf/96abmod.pdf.
80. T.J. Menzies and A. Mahidadia. Ripple-down rationality: A framework for maintaining psms. In *Workshop on Problem-Solving Methods for Knowledge-based Systems, IJCAI '97, August 23.*, 1997. Available from http://tim.menzies.com/pdf/97rdra.pdf.
81. M. Minsky. A framework for representing knowledge. In *The Psychology of Computer Vision*, 1975.
82. T.P. Moran and J.M. Carroll. *Design Rationale: Concepts, Techniques, and Use*. Lawerence Erlbaum Associates, 1996.
83. E. Motta. The knowledge modelling paradigm in knowledge engineering. In S.K. Chung, editor, *Handbook of Software and Knowledge Engineering (volume 1)*, 2001.
84. M. Mulholland, P. Preston, B. Hibbert, and P. Compton. An expert system for ion chromatography developed using machine learning and knowledge in context. In *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Edinburgh*, 1996.
85. NASA. CLIPS Reference Manual. Software Technology Branch, lyndon B. Johnson Space Center, 1991.
86. A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87–127, 1982.
87. A. Newell. Reflections on the Knowledge Level. *Artificial Intelligence*, 59:31–38, Feburary 1993.
88. N.J. Nilsson. Logic and artificial intelligence. *Artificial Intelligence*, 47:31–56, 1991.
89. B. Nuseibeh. To be *and* not to be: On managing inconsistency in software development. In *Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8)*, pages 164–169. IEEE CS Press., 1997.
90. J. Pearl and R.E. Korf. Search techniques. *Ann. Rev. Comput. Sci. 1987*, 2:451–67, 1987.
91. A. Pease, V. Chaudhri, F. Lehmann, and A. Farquhar. Practical knowledge representation and the darpa high performance knowledge bases project. In *Proceedings*

of KR-2000, 2000. Available from `http://reliant.teknowledge.com/HPKB/Publications/KR-195.ps`.

92. P. Preston, G. Edwards, and P. Compton. A 1600 Rule Expert System Without Knowledge Engineers. In J. Leibowitz, editor, *Second World Congress on Expert Systems*, 1993.
93. D. Richards and P. Compton. Combining formal concept analysis and ripple down rules to support the reuse of knowledge. In *SEKE '97: Proceedings of 1997 Conf. on Software Eng. & Knowledge Eng, Madrid*, 1997.
94. D. Richards and T.J. Menzies. Extending knowledge engineering to requirements engineering from multiple perspectives. In T.J. Menzies, D. Richards, and P. Compton, editors, *Third Australian Knowledge Acquisition Workshop, Perth*, 1997. Available from `http://tim.menzies.com/pdf/97akawre.pdf`.
95. D. Richards and T.J. Menzies. Extending the sisyphus iii experiment from a knowledge engineering task to a requirements engineering task. In *Banff Workshop on Knowledge Acquisition*, 1998. Available from `http://tim.menzies.com/pdf/98kawre.pdf`.
96. H.W.J. Rittel. Second generation design methods. In *Design Methods Group 5th Anniversary Report: DMG Occasional Paper, 1, 5-10.*, 1972. Also in N. Cross (Ed.) Developments in design methodology, (pp. 317-327). Chichester: Wiley & Sons, 1984.
97. P.S. Rosenbloom, J.E. Laird, and A. Newell. *The SOAR Papers*. The MIT Press, 1993.
98. D.A. Schon. *The Reflective Practioner*. Harper Collins/ Basic Books, 1983.
99. A. Th. Schreiber and W. P. Birmingham. The sisyphus-vt initiative. *International Journal of Human-Computer Studies*, 44(3/4), March/April 1996.
100. A. TH. Schreiber, B. Wielinga, J. M. Akkermans, W. Van De Velde, and R. de Hoog. Commonkads. a comprehensive methodology for kbs development. *IEEE Expert*, 9(6):28–37, 1994.
101. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *AAAI '92*, pages 440–446, 1992.
102. N. Shadbolt and K. O'Hara. Model-based expert systems and the explanations of expertise. In P.J. Feltovich, K.M. Ford, and R.R. Hoffman, editors, *Expertise in Context*, chapter 13, pages 315–337. MIT PRess, 1997.
103. N. Shadbolt, K. O'Hara, and L. Crow. The experimental evaluation of knowledge acquisition techniques and methods: History, problems and new directions. *International Journal of Human-Computer Studies*, 2000. (to appear).
104. H. Simon. *The Science of the Artificial*. MIT Press, 1969.
105. H.A. Simon. On the definition of the causal relationship. *Journal Philosphy*, 49:517–528, 1952.
106. L. Steels. Components of Expertise. *AI Magazine*, 11:29–49, 2 1990.
107. L. Suchman. Response to vera and simon's situated action: A symbolic interpretation. *Cognitive Science*, 17:71–75, 1993.
108. B. Swartout and Y. Gill. Flexible knowledge acquisition through explicit representation of knowledge roles. In *1996 AAAI Spring Symposium on Acquisition, Learning, and Demonstration: Automating Tasks for Users*, 1996.
109. G. Tecuci. *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. Academic Press, 1998.
110. J.D. Ullman. *Principles of Database and Knowledge-base Systems*. Computer Science Press, 1988.
111. F. van Harmelen and M. Aben. Structure-preserving specification languages for

knowledge-based systems. *International Journal of Human-Computer Studies*, 44:187–212, 1996.

112. F. van Harmelen and Annette ten Teije. Validation and verification of conceptual models of diagnosis. In *European Symposium on the Validation and Verification of Knowledge Based Systems, Leuven, Belgium*, 1997.
113. G. van Heust, A. Th. Schreiber, and B.J. Wielinga. Using explicit ontologies in kbs development. *International Journal of Human Computer Studies*, 45:183–292, 1997.
114. S. Waugh, T.J. Menzies, and S. Goss. Evaluating a qualitative reasoner. In Abdul Sattar, editor, *Advanced Topics in Artificial Intelligence: 10th Australian Joint Conference on AI*. Springer-Verlag, 1997. `http://www.cse.unsw.edu.au/~timm/pub/docs`.
115. G.I. Webb and J. Wells. Experimental evaluation of integrating machine learning with knowledge acquisition through direct interaction with domain experts. In *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop*, 1996.
116. B.J. Wielinga, A.T. Schreiber, and J.A. Breuker. KADS: a Modeling Approach to Knowledge Engineering. *Knowledge Acquisition*, 4:1–162, 1 1992.
117. B.C. Williams and J. DeKleer. Qualitative reasoning about physical systems: a return to roots. *Artificial Intelligence*, 51:1–9, 1991.
118. T. Winograd. Frame representations and the declarative/procedural controversy. In *Readings in Knowledge Representation*, pages 185–210. Morgan Kaufman, 1975. Also available R.J. Brachmann and H.J. Levesque (eds), Readings in Knowledge Representation, Morgan Kaufmann, Palo Alto,1985.
119. G. Yost. Implementing the Sisyphus-93 task using SOAR/TAQL. In B.R. Gaines and M. Musen, editors, *Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 46.1–46.22, 1994.
120. G.R. Yost and A. Newell. A Problem Space Approach to Expert System Specification. In *IJCAI '89*, pages 621–627, 1989.