(Submitted to the Journal of Software Engineering and Knowledge Engineering)

Issues with Meta-Knowledge

TIM MENZIES‡, KLAUS-DIETER ALTHOFF♀, YANNIS KALFOGLOU♦, ENRICO MOTTA♮
NASA/WVU Software Research Lab, 100 University Drive, Fairmont WV, USA, 26554
Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany
Institute for Representation and Reasoning; University of Edinburgh, UK
Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, UK
<tim@menzies.com,klaus-dieter.althoff@iese.fhg.de
yannisk@dai.ed.ac.uk,e.motta@open.ac.uk>

SUMMARY

At the SEKE'99 conference, knowledge engineering researchers held a panel on the merits of *meta-knowledge* **(i.e. problem solving methods and ontologies) for the development of knowledge-based systems. The original panel was framed as a debate on the merits of meta-knowledge for knowledge maintenance [21]. However, the debate quickly expanded. In the end, we were really discussing the merits of different technologies for the specification of reusable components for KBS. In this brief article we record some of the lively debate from that panel and the email exchanges it generated.**

Keywords: **Problem solving methods, ontologies, reuse, knowledge engineering, evaluation.**

What is meta-knowledge? How is it being used? Is that usage a useful thing to do? This article approaches these questions from our different viewpoints*:

The Constructors: Motta and Kalfoglou focus on the *construction* of knowledge base systems (KBS) using two special kinds of meta-knowledge: ontologies [14, 33] and problem solving methods (PSMs) [26, 6]. PSMs model the useful inference patterns seen in previous applications. Such patterns, it is argued, simplify and clarify future implementations. Ontologies model common domain terminology. This terminology might include the data structures required by a PSM. Using a good ontology, it is argued, can guide developers in the construction of new systems. For more information on ontologies, see `http://www.dai.ed.ac.uk/daidb/people/homes/yannisk/seke99panelhtml.html`.

The Maintainers: Althoff and Menzies focus on the *maintenance* and modification of KBS using case-based reasoning (CBR) [3, 1] and continual testing [10, 22]. CBR researchers argue that people rarely solve problems from scratch. Rather, they find similar old solutions, then adapt them to new situations. Continual testing is a reflective approach to design; i.e. improvements are prompted by some device reporting errors [30, 7]. The most important component of reflective design is the evaluation device since failed evaluations are essential for improving the design.

* For more details on these views, see our position papers in the proceedings of SEKE'99

© 1999

Received September 13, 1999
Revised Whenever

Construction vs maintenance is only an approximation of our different positions. Clearly, the two approaches overlap. Constructors believe that maintenance can be simplified via use of good ontologies and problem solving methods. Maintainers believe that their "maintenance" tools can really be used as construction tools since any maintenance task involves some (re)construction of parts of a system.

The following list of reported benefits for meta-knowledge shows specific areas where we agree and disagree. We all agree on the *communications, interoperability* and *browsing/searching* benefits. However, while the constructors endorse the *guidance* and *systems engineering* benefits, the maintainers do not.

The communications benefit: Any communication task is simplified by a shared lexicon. Ontologies are a very useful collection of knowledge for systematically viewing and sharing a specific topic/problem. For example, ontologies provide a unified framework within an organization that reduces the terminological confusion[34] arising from different contexts and viewpoints for a particular domain. Meta-knowledge is also useful for any teaching situation. For example, meta-knowledge can simplify a student's task when (e.g.) reading textbooks. Also, Menzies reports that abstract meta-knowledge descriptions are useful when tutoring software or knowledge engineering [24]. Such abstractions serve as a useful final initiation ritual for a novice designer. When they "get" abstractions, we know that the students are capable of comparing and contrasting a wide range of systems.

The interoperability benefit: Inter-operability among systems with different modeling methods, paradigms, languages and software tools can be achieved with ontologies that act as an inter-lingua [16].

The browsing/searching benefit: The meta-knowledge within an ontology can assist an intelligent search engine with processing a query. For example, if a query returns no results, then the ontology could be used to automatically generalize the query to find nearest partial matches.

The systems engineering benefit: Ontologies and PSMs, it is argued, simplifies system construction. For example, Kalfoglou executes the constraints found in existing ontologies to check new systems. Such pre-existing constraints, are a powerful tool for checking knowledge when other oracles are absent [14]. In other work, one commercial company used the ontology associated with Motta's PSM design tools to formalize the regulations applicable to the design of the truck cabin. This formalization, associated with a constraint analyzer, cut the design of the geometric layout of the cabin from 4 months to 1 day(!!). Elsewhere, an intelligent PSM librarian was used to build nine KBS applications. Development times changed from one to 17 days (using the librarian) to a range of 63 to 250 days (without using the librarian) [19]. Finally, as a last example, we mention the SALT KBS editor used for the VT elevator configuration system. SALT restricted its knowledge editors to only those terms relevant for the propose-and-revise PSM used in VT [18, 17]. [illegible] of VT's rules could be auto-generated by SALT.

The guidance benefit: While we may use little of an ontology or a PSM, it may still be useful as a "pointer tool". That is, the ontology/PSM could be used as a structuring tool for exploring a new domain. Roughly speaking, reusing abstracted forms of old knowledge is pointing the way saying "these kinds of things are important, even if these particular things are not". In this approach, developers kick-start the development with an ontology/PSM.

Motta argues strongly for the *systems engineering* and *guidance* benefits. PSM research, he says, has transformed knowledge engineering from "an art" [13] to a structured discipline, organized in terms of a number of generic problem areas (e.g., diagnosis, planning, etc...). The PSM-literate KBS developers can worry less about issues such as (e.g.) conflict resolution strategies for production systems. Such

low-level details do not reflect the knowledge-level goals of a domain expert [27, 8]. In a knowledge-level analysis, the task is to map the space of knowledge engineering techniques and develop handbooks similar to those of other engineering disciplines. These handbooks specify the type of problems we deal with and the kind of techniques we use. In Motta's view PSMs and ontologies provide the key technologies for writing these handbooks.

Menzies reply to Motta is to note that, despite arguments like those made by Motta and Kalfoglou and their colleagues, PSMs are more widely used in Europe than elsewhere. Many others approaches exist [20] and some can claim significant successes. For example:

- The United States DARPA High Performance Knowledge Based initiative (HPKB) studied how to write KBs faster [23]. 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 [32]. DISCIPLE includes machine learning tools for abstracting learnt rules which makes them more generally applicable. As DISCIPLE runs, its builds and updates the meta-knowledge used for the purposes of abstraction.
  Proponents of randomized search algorithms do not use PSM-style structuring tools for organizing their knowledge bases. Instead, they model their entire domains in simple disjunctive normal form (DNF). In comparative studies, such randomized search over 3DNF has out-performed knowledge-intensive approaches by 2 to 3 orders of magnitude [31].
  In the RDR approach, a KBS stores a patch tree to a knowledge base (each patch fixes a rule and may itself patched recursively). Candidates for new patches are inferred by an analysis of the paths taken through the KB and the patches found on those paths. Patch histories are low-level syntactic knowledge yet capture the context of change of an expert system. Very large expert systems have been built and maintained in this manner, without needing knowledge engineers [10, 11, 28, 29].

Althoff and Menzies have other doubts about *systems engineering* and the *guidance* benefits using PSMs and ontologies:

> It may not be cost-effective to routinely devoting a significant portion of the systems engineering effort to formalizing most of the domain knowledge. Althoff argues that enshrining domain terminology (e.g. into an ontology) should be the exception, rather than the rule. Over time, the evaluation of some experience base will show which artifacts will be the most useful ones (and only these useful artifacts should be enshrined).
> Also, the level of abstraction at which we formalize domain terms should be learnt via extensive experience with that particular term [4]. For example, sometimes, simple text-based descriptions of meta-knowledge are powerful tools. Object-oriented "guidance patterns" serve to direct the analyst's focus onto a set of issues that previous analysts have found insightful. Such patterns include CHECKS [12], Caterpillar's Fate [15], and Coad's strategies [9]. This type of meta-knowledge is stored as simple checklists of English text.
> The systems engineering benefits described above assumed some reuse of ontologies and PSMs. The software engineering experience is that reuse comes at a cost. Verbatim reuse is very rare: often artifacts from a reuse library have to be modified prior to their use. The COCOMO-II software cost estimation model offers an estimate of the cost of adapting reusable sub-routines for a new project [2]. A learning curve must be traversed before a module can be adapted. According to COCOMO-II, 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 11 in Menzies' panel notes[21]). The COCOMO-II results relate to the adaption cost of procedural systems. Hence, they may not apply to declarative descriptions of system terminology (i.e. an ontology). However, at the very

least, these results caution us that just because we are reusing an ontology or a PSM, this does not necessarily mean that we are building systems more cheaply. Ontologies and PSMs must be learnt prior to use and this learning time may have a non-trivial impact on the overall cost.

It can be argued that that the last point must be false. If it were true, then why can we find the spectacular optimizations of systems development using PSMs and ontologies seen above (recall the examples in the *systems engineering* sections)? Menzies objects to this counter-argument, arguing that without more precise metrics collection, it is hard to interpret these reported optimizations [20, 23] (e.g. when Motta's colleagues reduced their design time from 4 months to 1 day, how much of that reduction was due to the PSM framework and how much to the constraint analyzer that they used?).

Another point of contention was the stability of the meta-knowledge:

- Motta defines meta-knowledge as that knowledge which is stable across domains. Stable knowledge does exist, he says and argues by example: "You do not change the C compiler every time you change the C code". For many PSMs such as diagnosis: there exist a finite number of different approaches to diagnosis, each one with pros and cons [5].
  Menzies took the opposite view. Unlike Motta, he doubts that any detailed description of meta-knowledge will ever be stable. Meta-knowledge is not useful if it simplifies system construction but complicates system maintenance. Meta-knowledge is still knowledge, he says, and the maintenance of system knowledge is a significant cost of any successful system. Further, he rejects the diagnosis example saying that his reading of the details of the diagnosis literature is that separate and incompatiable abstractions are offered by different authors [24]. More generally, based on a recent literature review [25] he claims that in the usual case, expert use of a KBS produces significant changes to that KBS.

In the end, what did we learn? For one thing, we now have a clearly definition of where we agree and disagree. For another, we now have an explicit list storing the potential benefits of ontologies and PSMs (to the best of our knowledge, this list was only implicit in the current literature). Lastly, the participants learnt much about each other's positions which would be useful for future debates (at SEKE 2000?). Based on a flurry of email, we know have guidelines on what arguments would win over the other side:

> We need more KE metrics. In particular, we need to evaluate the success (or otherwise) of different parts of our systems. Also, we need to test for the stability of our meta-knowledge.
> The constructors/maintainers have to be more explicit about their maintenance/construction methods respectively.

REFERENCES

1. A. Aamodt and E. Plaza. Case-based reasoning; Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, March 1994. Available from `http://www.iiia.csic.es/People/enric/AICom_ToC.html`.
2. 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`.
3. K-D. Althoff, A. Birk, S. Hartkopf, W. Muller, M. Nick, D. Surmann, and C. Tautz. Managing Software Engineering Experience for Comprehensive Reuse. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslauten, Germany*, pages 10–19, June 1999.

4. K.-D. Althoff, M. Nick, and C. Tautz. Improving Organizational Memories Through User Feedback. In F. Bomarius, editor, *Proc. of the Workshop on Learning Software Organizations (LSO) (in conjunction with the11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslauten, Germany)*, pages 27–44, June 1999.
5. R. Benjamins. *Problem Solving Methods for Diagnosis*. PhD thesis, University of Amsterdam, 1993.
6. R. Benjamins and D. Fensel. Special Issue on Problem Solving Methods. *International Journal of Human Computer Studies*, 49(4), 1998.
7. 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.
8. W. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.
9. P. Coad, D. North, and M. Mayfield. *Object Models: Strategies, Patterns, and Applications*. Prentice Hall, 1997.
10. 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.
11. P.J. Compton and R. Jansen. A Philosophical Basis for Knowledge Acquisition. *Knowledge Acquisition*, 2:241–257, 1990.
12. W. Cunningham. The CHECKS Pattern Language of Information Integrity. In J. Coplien and D. Schmidt, editors, *Pattern Languages of Program Design*. Addison-Wesley, 1995. Also avialable at *http://c2.com/ppr/checks.html*.
13. E. A. Feigenbaum. The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering. In *IJCAI '77*, 1977.
14. Kalfoglou,Y. and Robertson,D. A Case Study in Applying Ontologies to Augment and Reason about the Correctness of Specifications. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslauten, Germany*, pages 64–71, June 1999.
15. N. Kerth. Caterpillar's Fate: A Pattern Language for Transformation from Analysis to Design. In J. Coplien and D. Schmidt, editors, *Pattern Languages of Program Design*. Addison-Wesley, 1995. Also available from *http://c2.com/ppr/catsfate.html*.
16. J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, G Yost, and other members of the PIF working group. The PIF Process Interchange Format and framework. *Knowledge Engineering Review*, 13(1):91–120, February 1998.
17. S. Marcus and J. McDermott. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39:1–37, 1 1989.
18. S. Marcus, J. Stout, and J. McDermott. VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking. *AI Magazine*, pages 41–58, Winter 1987.
19. 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.
20. T. Menzies. hQkb- The High Quality Knowledge Base Initiative (Sisyphus V: Learning Design Assessment Knowledge). In *KAW'99: the 12th Workshop on Knowledge Acquisition, Modeling and Management, Voyager Inn, Banff, Alberta, Canada Oct 16-22, 1999*, 1999. Available from `http://www.csee.wvu.edu/~timm/docs/9905hqkb0.html`.
21. T. Menzies. Knowledge Maintenance Heresies: Meta-Knowledge Complicates KM. In *11th Annual International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, June 17 - 19, 1999*, 1999. Available from `http://research.ivv.nasa.gov/docs/techreports/1999/NASA-IVV-99-008.pdf`.
22. T. Menzies and C.C. Michael. Fewer Slices of PIE: Optimising Mutation Testing via Abduction. In *SEKE '99, June 17-19, Kaiserslautern, Germany. Available from `http://research.ivv.nasa.gov/docs/techreports/1999/NASA-IVV-99-007.pdf`*, 1999.
23. 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*, 2000. forthcoming; avaiable from `http://www.csee.wvu.edu/~timm/docs/eke.pdf`.
24. T.J. Menzies. OO Patterns: Lessons from Expert Systems. *Software Practice & Experience*, 27(12):1457–1478, December 1997. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/97probspatt.ps.gz`.
25. T.J. Menzies. Towards Situated Knowledge Acquisition. *International Journal of Human-Computer Studies*, 49:867–893, 1998. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/`

98ijhcs.
26. E. Motta, D. Fensel, M. Gaspari, and A. Benjamins. Specifications of Knowledge Components for Reuse. In *Proceedings of SEKE '99*, 1999.
27. A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87–127, 1982.
28. 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.
29. 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.
30. D.A. Schon. *The Reflective Practioner*. Harper Collins/ Basic Books, 1983.
31. B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *AAAI '92*, pages 440–446, 1992.
32. G. Tecuci. *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. Academic Press, 1998.
33. M. Uschold and M. Gruninger. Ontologies: Principles, Methods, and Applications. *The Knowledge Engineering Review*, 11(2):93–136, 1996.
34. M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13(1), February 1998.