Knowledge Maintenance: The State of the Art

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

June 1, 1999

What is KM?

Knowledge maintenance (KM) of knowledge-based systems =

- Insight + reflect + change + preserve

Insight = seeing bug or enhancements
Reflect = searching the dependencies between concepts
Change = the easiest thing to do badly, the hardest thing to do correctly.
Preserve = changes that address new insights do not introduce bugs into older concepts.

What is not KM?

An ability to rapidly build or change a system:

- The “rapid acquire assumption”.
- Insufficient unless “build” or “change” supports “preserve”.

An ability to watch knowledge execute:

- The “operationalisation KM assumption”.
- Insufficient without “behavioural knowledge” (see below)

An ability to check a KB using meta-knowledge:

- The “recursive maintenance problem”.
- Insufficient unless the meta-knowledge can be maintained as well.

Tutorial: summary #1

- Survey of KM techniques
- Challenge issues for KM research (see page 2).
- A call-to-arms: certain incorrect assumptions in modern KM research.
Tutorial: summary #2

Trite sound bites summarizing my views:

- Knowledge does not stabilise.
- Experts don’t agree.
- Modeling is ok, but models aren’t.
  - Useful: building models
  - Useless: carving them in stone
- Reuse:
  - Likely: reuse=productivity=myth.
  - Unlikely: me reuse your ideas.
- Maintain the process, not the product.

A Quick KM Primer

KM state-of-the-art [Menzies, 1999] (see this reference for more details on the material in this tutorial).

KM tools supporting preserve:

- Logic-based: [Debenham, 1998, Compton and Jansen, 1990]

Meta-knowledge to support KM: [Bachant and McDermott, 1984, Marcus et al., 1987]

KM without meta-knowledge: [Richards and Compton, 1997]


General texts:

- Empirical artificial intelligence [Cohen, 1995]
- Advice from SE [Fenton and Pfleeger, 1997]

Preliminaries

<table>
<thead>
<tr>
<th>Preliminaries</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive Summary</td>
<td>1-5</td>
</tr>
<tr>
<td>Objective</td>
<td>6</td>
</tr>
<tr>
<td>Brief Description</td>
<td></td>
</tr>
<tr>
<td>Audience</td>
<td></td>
</tr>
<tr>
<td>Expectation management</td>
<td></td>
</tr>
<tr>
<td>About the author</td>
<td></td>
</tr>
<tr>
<td>Author biases</td>
<td></td>
</tr>
<tr>
<td>Motivation</td>
<td>Change the only constant</td>
</tr>
<tr>
<td>The KM space</td>
<td>Types of knowledge</td>
</tr>
<tr>
<td>Discussion</td>
<td>How to commission a new KM tool</td>
</tr>
<tr>
<td>References</td>
<td>83</td>
</tr>
</tbody>
</table>
In the software and knowledge engineering literature we can see maintenance strategies offered to maintain seven main types of knowledge and five main ways these are processed.

This tutorial will review dozen of software systems that contribute to these $7 \times 5 = 35$ types of knowledge maintenance to make the following conclusions:

- Open issues with the current maintenance research are identified. These include (a) areas that are not being addressed by any researcher; (b) the “recursive maintenance problem”; and (c) drawbacks with “rapid acquire systems” and the “operationalisation KM assumption”.

- A process is described for commissioning a new maintenance tool.

- A general common principle for maintenance (search-space reflection) is isolated.

Practicing software engineers interested in using or assessing leading edge technologies; software managers reviewing the state of the art in their field; software engineering and knowledge engineering researchers; graduate students preparing literature reviews of their field.

Dr. Tim Menzies holds a Ph.D. in artificial intelligence from the University of New South Wales, Australia, and works at the NASA/WVU Software Independent Verification and Validation Facility, Fairmont West Virginia. In that position, he explores logical methods for improving our ability to reason about the products of software engineering (specs, code, design documents).

At the time of this writing, Dr. Menzies has 67 publications, dozens of which are in international refereed forums. Much of this work is an analysis of where our current generation of software engineering and knowledge engineering techniques stop working.

He is also an active figure in the knowledge acquisition (KA) community and is one of the co-chairs of the evaluation of knowledge engineering methodologies track at the annual international KA workshop.

You skipped lots of slides.” Can’t cover all this material in 3 hours.

Some material was rushed” or “I wanted more details on XYZ” This is an overview only:

- Every page is someone’s research life- can’t show it all.
- If we pique(*) your interest in XYZ, and you know XYZ is connected to ABC, we have succeeded.
- For more details, see [Menzies, 1999].

(*)Pique, v. t. To excite to action by causing resentment or jealousy; to stimulate; to prick; as, to pique ambition, or curiosity.
Biases of the Author

The majority Euro-view of KA/KM:

- Understand and adapt KBs via meta-knowledge of cliched terms and inference procedures (a.k.a. ontologies and problem solving methods).
- e.g. KADS [Wielinga et al., 1992a]

The minority Australian-view of KA/KM (includes author):

- Meta-knowledge only delays the KM problem (how do you maintain the meta-K?).
- Alternative: maintain via syntactic structures such as context (RDR) or dependency graphs (knowledge normalization).
- e.g. RDR, p60 [Preston et al., 1993,Richards and Compton, 1997,Menzies, 1998];
- e.g. knowledge normalization,p 51 [Debenham, 1998];

True or False?

The explicit and high-level expression of knowledge in a KBS makes them easy to maintain.

False!

XCON:

- Half of its thousands of rules changed every year [Soloway et al., 1987].
- ?? due to changing environment (XCON configured computers for DEC computers and DEC keeps releasing new computers).

Garvin ES-1:

- KBS change occurs in even static domains [Compton et al., 1989]

![Graph](image-url)
A Garvin ES-1 Rule (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")

Experts Disagreeing

[Shaw, 1988]: used repertory grids to compare the meaning of terms used by different experts on a common geology problem.

A. The calibrating experiment: 12 weeks later, self-review. These figures given baseline expected agreement figures for this instrument.

<table>
<thead>
<tr>
<th>Expert</th>
<th>%understands</th>
<th>%agrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>E₁</td>
<td>62.5</td>
<td>81.2</td>
</tr>
<tr>
<td>E₂</td>
<td>77.8</td>
<td>94.4</td>
</tr>
<tr>
<td>E₃</td>
<td>85.7</td>
<td>78.6</td>
</tr>
</tbody>
</table>

B. The real experiment. Note E₁,E₃: very low levels of agreement.

<table>
<thead>
<tr>
<th>Expertpairs</th>
<th>%understands</th>
<th>%agrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>E₁,E₂</td>
<td>62.5</td>
<td>33.3</td>
</tr>
<tr>
<td>E₂,E₃</td>
<td>61.1</td>
<td>26.7</td>
</tr>
<tr>
<td>E₁,E₃</td>
<td>31.2</td>
<td>8.3</td>
</tr>
<tr>
<td>E₂,E₃</td>
<td>42.9</td>
<td>33.3</td>
</tr>
<tr>
<td>E₁,E₃</td>
<td>44.4</td>
<td>20.0</td>
</tr>
<tr>
<td>E₂,E₃</td>
<td>71.4</td>
<td>33.3</td>
</tr>
</tbody>
</table>

Does Knowledge Stabilize?

[Catlett, 1991]: use C4.5 [Quinlan, 1986] to learn decision trees for 20 problems using either:
- all the N=3000..5000 training cases or
- half the cases (randomly selected)

In all but 1 case (demon, first row), more experience meant significantly less errors, but larger theories,

<table>
<thead>
<tr>
<th>domain</th>
<th>Δ tree size</th>
<th>Δ error</th>
</tr>
</thead>
<tbody>
<tr>
<td>demon</td>
<td>0.97</td>
<td>0.51</td>
</tr>
<tr>
<td>wave</td>
<td>1.91</td>
<td>0.95</td>
</tr>
<tr>
<td>diff</td>
<td>1.46</td>
<td>0.69</td>
</tr>
<tr>
<td>othello</td>
<td>1.68</td>
<td>0.8</td>
</tr>
<tr>
<td>heart</td>
<td>1.61</td>
<td>0.65</td>
</tr>
<tr>
<td>sleep</td>
<td>1.73</td>
<td>0.91</td>
</tr>
<tr>
<td>hyper</td>
<td>1.74</td>
<td>0.83</td>
</tr>
<tr>
<td>hypo</td>
<td>1.45</td>
<td>0.85</td>
</tr>
<tr>
<td>binding</td>
<td>1.51</td>
<td>0.82</td>
</tr>
<tr>
<td>replace</td>
<td>1.38</td>
<td>0.8</td>
</tr>
<tr>
<td>euthy</td>
<td>1.33</td>
<td>0.61</td>
</tr>
<tr>
<td>mean</td>
<td>1.52</td>
<td>0.77</td>
</tr>
</tbody>
</table>
Does Knowledge Stabilize? (2)

Situated cognition:

- Using knowledge changes knowledge [Menzies and Clancey, 1998].

For example, [Shalin et al., 1997]:

- Experts do modify their behaviour according to community standards of “accepted practice”.
- But it’s only novices who slavishly re-apply that accepted practice.
- Experts adapt accepted practice when they apply it:
  - Partially match current problem to libraries of accepted practice.
  - Implement an acceptance test for their adaptation.
  - Modify the accepted practice library if acceptance failure.

Reuse & Productivity

Q: Is old knowledge (that we can adapt) still a productivity tool for new apps?

A1: YES if adaptations small (< 1.2.5%). Otherwise, overall cost benefits not impressive. Data from 2954 NASA SE modules [Abts et al., 1998, p21].

A2: Maybe not. Decades-old diagnosis models give same production benefits as models invented very quickly. And no model gives higher production benefits. Data from KA experts reading patient-doctor transcripts [Corbridge et al., 1995]

So, What Role for Models?

My argument is:

- Building new models is useful:
  - Unites a community.
  - Reflection over models let us plan for events we have not/cannot directly experience.

- Holding on to old models may not be useful:
  - While I may reuse some of my old ideas;
  - I doubt that I will reuse yours we belong to the same clan.

<table>
<thead>
<tr>
<th>Reuse Model</th>
<th>% disorders identified</th>
<th>% knowledge fragments identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straw man: invented very quickly</td>
<td>50</td>
<td>28</td>
</tr>
<tr>
<td>KADS diagnosis model: decades of work</td>
<td>55</td>
<td>34</td>
</tr>
<tr>
<td>No model</td>
<td>75</td>
<td>41</td>
</tr>
</tbody>
</table>
KM is here to stay.

Not because of:

- Sloppy analysis
- Lack of formal rigor (e.g., formal methods)
- Poor management practices
- The wrong tools
- etc etc

But because of:

- The fundamental nature of expert knowledge.
- Using "it" changes "it".
- Hitting the nail changes your grip on the hammer.
- Exercising knowledge makes you refine that knowledge

Pressing and urgent issue: how to improve KM?

The KM Space

In the literature, I can see 7 kinds of knowledge being maintained using 5 activities [Menzies, 1999].

<table>
<thead>
<tr>
<th>Ktype: Knowledge type</th>
<th>Ptype: Processing Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acquire</td>
</tr>
<tr>
<td>WordK</td>
<td>all</td>
</tr>
<tr>
<td>SentenceK (ontologies)</td>
<td>most</td>
</tr>
<tr>
<td>BehaviouralK</td>
<td>some</td>
</tr>
<tr>
<td>PSMs</td>
<td>some</td>
</tr>
<tr>
<td>FixK</td>
<td>few</td>
</tr>
<tr>
<td>SocialK</td>
<td>few</td>
</tr>
</tbody>
</table>

Comments:

- KM needs knowledge acquisition (KA). To maintain "it", first you have to get "it". Next you might have to update "it".
- Not all parts of the 7*5 points in the KM space are covered.
- Large space to explore. High points only at this tutorial. See [Menzies, 1999] for more details.
- Must explain the Ktypes before the Ptypes.

7 Kinds of Knowledge

<table>
<thead>
<tr>
<th>Valid</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>1-5</td>
</tr>
<tr>
<td>✓</td>
<td>5</td>
</tr>
<tr>
<td>✓</td>
<td>13</td>
</tr>
<tr>
<td>✓</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>49</td>
</tr>
<tr>
<td>✓</td>
<td>80</td>
</tr>
</tbody>
</table>

References

K-type: WordK

WordK: the things we can’t divide any further. E.g.

- Logic programming: wordK=propositions
- OO high-level design wordK=method names
- Functional systems: word=simple functions (e.g. a comparison age(patient)=old).
- Rule-based systems: wordK=atoms; e.g.:
  
if infection is meningitis and infection is bacterial and patient has undergone surgery and the patient has had neurosurgery and neurosurgery-time was < 2 months ago and patient got a ventricular-urethral-shunt

then infection = eColi (.8) or klebsiella (.75)

[Clancey, 1992]

K-type: BehaviouralK

Knowledge of known or desired or past behaviour.

BehaviouralK in case-based reasoning [Kolodner, 1991, Kolodner, 1993]: new situations are managed by retrieving and modifying previously see cases.

K-type: BehaviouralK & HT4 (2)

[Menzies and Compton, 1997]

A hypothetical economics theory

world W.1

world W.2

BehaviouralK in case-based reasoning [Kolodner, 1991, Kolodner, 1993]: new situations are managed by retrieving and modifying previously see cases.

K-type: SentenceK

The rule on page 28 is sentenceK connecting that connect wordKs.

Meta-knowledge of legal sentence types=ontology: “an explicit specification of a conceptualization” [Gruber, 1993]. E.g. here is an ontology of design discussions [Ramesh and Dhar, 1992].

In an ontology, abstract terms often appear high in some isa hierarchy while specific domain terms appear lower down the hierarchy; e.g. recall the 7 types of knowledge on page 27.
K-type: BehaviouralK (3)

Coverage tools = SE behaviouralK

- All branches coverage
- All usages coverage
- All decision points coverage
- All ... coverage

Desired behaviour specified as global constraints:

- Succinct expression
- Can be checked directly using model-checkers [Clarke et al., 1986]
- Can be checked via theorem-provers: ground, then negate constraints. Error detected if the negated constraints are reachable.

K-type: PSMs: Problem Solving Methods

[Clancey, 1992]. Recall this rule:

if infection is meningitis and infection is bacterial and patient has undergone surgery and the patient has had neurosurgery and neurosurgery-time was < 2 months ago and patient got a ventricular-urethral-shunt
then infection = eColi (.8) or klebsiella (.75)

This rule blurs the true heuristic ...

if patient got a ventricular-urethral-shunt
then infection = eColi (.8) or klebsiella (.75)

... with more general problem solving method knowledge:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exploreAndRefine</td>
<td>Explore super-types before sub-types.</td>
</tr>
<tr>
<td>findOut</td>
<td>If an hypothesis is subsumed by other findings which are not present in this case then that hypothesis is wrong.</td>
</tr>
<tr>
<td>testHypothesis</td>
<td>test causal connections before mere circumstantial evidence.</td>
</tr>
</tbody>
</table>

Operationalisation KM Assumption

"If we can watch a program execute, then we can understand it."

Definitions:

- Test cases : <input, output>
- Anomaly: expected(output) ≠ actual(output)
- Mere program running: <input, expected(output) = 0>, i.e. no behaviouralK.

Output can be voluminous, too slow to process (HT4=NP hard): need heuristic agents to find significant outputs

- Compare with desired (HT4- page 31)
- Code coverage
- Model checkers

DON'T ask the developer to evaluate via program watching:

- "Halo effect"
- "What we need is not opinions or impressions, but relatively objective measures of performance." [Cohen, 1995, p74].

Ktype: PSMs (2)

Inside a medical diagnosis system...

...and a fault system for electrical circuits...

... is the same inference cliche: heuristic classification [Clancey, 1985].
Ktype - PSMs (3)

Solo PSM:

Multiple PSMs:
- SPAR/BURN/FIREFIGHTER (SBF) [Marques et al., 1992];
- generic tasks [Chandrasekaran et al., 1992]; configurable role-limiting methods [Swartout and Gill, 1996, Gil and Melz, 1996]; model construction operators [Clancey, 1992];
- CommonKADS [Wielinga et al., 1992a, Schreiber et al., 1994]; the Method-To-Task approach [Eriksson et al., 1995];
- components of expertise [Steels, 1990]; MIKE [Angele et al., 1996];
- PSM approaches contrasted in the Related Work section of [Wielinga et al., 1992a].

Ktype: PSMs (4): success stories

SALT KB editor for VT (elevator configuration):
- PSM meta-knowledge restricted the dialogues to only those terms relevant for the propose-and-revise PSM used in VT [Marcus and McDermott, 1989].
- (2130/3062 \approx 70\%) of VT’s rules were auto-generated by SALT.

SPAR/BURN/FIREFIGHTER [Marques et al., 1992]:
- 9 applications (hardware configuration).
- Intelligent editor to explore distinguishing features between PSMs.
- Auto-configuration of executable from PSM library.
- Development times = one to 17 days (using SBF).
- Development times = to 63 to 250 days (without using SBF).
- High-water mark in reported productivity increases in software or knowledge engineering.

Digression: PSMs, Ontologies, and Patterns

[Menzie, 1997] argues that OO patterns of design [Gamma et al., 1995] and architecture [Buschmann et al., 1996, Shaw and Garlan, 1996] are analogous to PSMs and ontologies:
- Both recent abstract descriptions of supposedly common parts of many designs.
- OO patterns are typically structural; exceptions: some of the patterns in [Fowler, 1997].
- PSMs patterns are typically functional (recall page 35).
- Ontologies are very compatible with the patterns literature.

KA has building PSMs since at least 1983 [Chandrasekaran, 1983] and ontologies [Neches et al., 1991] since at least 1991:
- Current problems with PSMs/ontologies will be encountered in the future by patterns research.

Problems with PSMs

(Caution: minority view, recall page 12.)

PSMs are not stable over time:
- PSM primitives offered by Clancey [Clancey, 1992], KADS [Wielinga et al., 1992b], and SBF are different.

The number and nature of the PSMs not fixed. Often, new domain = new PSMs [Linster and Musen, 1992].
- Diagnosis PSM, has not stabilized, may not do so in the near future:
  - [Menzie, 1997] describes eight different models of diagnosis (four from the problem solving method community, four from elsewhere).
  - Some common features,
  - But, they reflect fundamentally divergent different views on how to perform diagnosis.

Ditto for ontologies (recall Cohen results on page 21 - reuse results not impressive).
Ktype: QualityK

QualityK = some manner of generating an opinion about the value of the KB.

E.G. using behaviouralK:
- HT4 qualityK (page 31)- a good theory can generate worlds containing known or desired behaviour

E.G. using non-functional requirements:
- e.g. portability, evolvability, development affordability, security, privacy, or reusability:

...[diagram]

[Boehm, 1996]

Ktype: QualityK (2) via Critical Success Metrics (CSMs)

[Menzies, 2000]

Domain-specific; obvious in retrospect (can take weeks of analysis to uncover) may require extra data capture; reflect the contribution of the software to a particular business context, hence:
- Typically do not refer to internal properties of a program.
- Cannot be developed by programmers without from business users.
- Can only be collected once the program is running in its target context.

E.G. CSMs for a pig-farm management system [Menzies et al., 1992]:
- NOT the European vs Australian protein unification model
- Rather, profit per M² per day

...[graph]

Ktype: QualityK (3) via Product-Oriented Assessment

House of quality [Shaw and Garlan, 1996].

Syntactic anomaly knowledge [Preece and Shinghal, 1992]:
- Redundancy
- Redundant rules
- Duplicate rules
- Subsumed rules
- Ambivalence
- Conflicting sets of rules
- Circularity (inference loops)
- Deficiency
- Missing rules
- Missing values

And, by the way, fielded expert systems have these anomalies [Preece and Shinghal, 1992] (Errors/Anomalies):

<table>
<thead>
<tr>
<th></th>
<th>MMU</th>
<th>TAPES</th>
<th>NEURON</th>
<th>DISPLAN</th>
<th>DMST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (literals)</td>
<td>105</td>
<td>150</td>
<td>190</td>
<td>350</td>
<td>540</td>
</tr>
<tr>
<td>Logical subsumption errors</td>
<td>0</td>
<td>7.5</td>
<td>0</td>
<td>4.9</td>
<td>5</td>
</tr>
<tr>
<td>Missing rule errors</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>17/59</td>
<td>0</td>
</tr>
<tr>
<td>Circularities in reasoning errors</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20/24</td>
<td>0</td>
</tr>
</tbody>
</table>

ETM: a Product-Oriented Assessment Tool

ETM= the EXPECT transactions manager [Gil and Tallis, 1997].
- Detects errors in PSMs in a LOOM representation via a partial evaluation of methods.
  - This partial evaluation is driven by a particular example (a.k.a. an item in the behaviouralK).
- Errors are detected if a method cannot fire because the types of the input parameters to the methods are not available (formally, this is a variant on PSB “unusable rules” from page 42).
- The completeness of this error detector is a function of the completeness of the behaviouralK used to drive the partial evaluator.
What is a conflict?

- General answer: \( \text{oops}(X, \text{not } X) \).

Other answers for different representations:

- HT4 (page 31) variables have one and one only state assignment:

  \[
  \text{oops}(	ext{Var1/State1}, \text{Var1/State2}) :- \not \text{(State1 = State2)}.
  \]

- Class hierarchies: sub-class constraints should be violated by a super-class (the substitution principle).

- State-transition diagrams from 2 authors. Conflicts if [East-erbrook and Nuseibeh, 1996];
  - A transition exists between two states in one diagram;
  - Those two states appear in the other diagram;
  - The transition does not appear in the other diagram.

InconsistencyK for WordK using repertory grids [Gaines and Shaw, 1989]

Consensus: same item, same categorisations;

Correspondence: (a.k.a. synonyms) items with different names, but the same categorisation;

True conflict: same items, different categorisations;

Contrast: different items, different categorisations

Knowledge cannot be understood with understanding its social context [Winograd and Flores, 1987, Clancey et al., 1996].

Paper approaches that informally describe the organizational context of a system:

- The organizational model of KADS [Wielinga et al., 1992b]
- The stakeholders of the Olle-126 [Olle et al., 1991].

Operationalized socialK:

- Clancey et al.'s BRAHMS system [Clancey et al., 1996]: macro-workflow of an organization is an emergent process that is inferred from all the micro-behaviour of the agents in an organization.

- Design rationales are a record of why a community decided to change some aspect of a system [Moran and Carroll, 1996]. E.g. REMAP (page 29):
  - Logs design discussions.
  - Can track the impact of a change of mind to the constraints on the development.
  - Can replay previous discussions to generate historical understanding of how some decision was achieved.
The Recursive Maintenance Problem

Who shaves the barber? Who guards the guardians? How do we maintain the maintenance knowledge (QualityK, FixK, SocialK, Ontologies, PSMs)?

If we use KB2 to build and assess KB1, do we need KB3 to build KB2 and KB4 to build KB3 and . . . ?

Theory: don’t need to maintain KB2. So succinct, errors obvious. Wrong!

Results from the Sisyphus-II experiments (re-implement VT using your favorite KE approach). 13/25 elevator configurations failed due to basic flaw in all the Sisyphus-II implementations' [Zdrahal and Motta, 1996]. This problem was reported in only one of the other Sisyphus-II contributions:

* Sisyphus-II propose-and-revise is a local greedy search. Local hill-climbing may ignore solutions which are initially unpromising, but lead later on to better solutions.

5 Processing Activities

<table>
<thead>
<tr>
<th>Ktype: Knowledge type</th>
<th>Acquire</th>
<th>Operation-</th>
<th>Fault</th>
<th>Fix</th>
<th>Preserve</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordK</td>
<td>all</td>
<td>most</td>
<td>few</td>
<td>few</td>
<td>few</td>
</tr>
<tr>
<td>SentenceK (ontologies)</td>
<td>most</td>
<td>most</td>
<td>many</td>
<td>many</td>
<td>few</td>
</tr>
<tr>
<td>BehaviouraK</td>
<td>some</td>
<td>few</td>
<td>many</td>
<td>few</td>
<td>none</td>
</tr>
<tr>
<td>PSMs</td>
<td>some</td>
<td>many</td>
<td>few</td>
<td>few</td>
<td>few</td>
</tr>
<tr>
<td>QualityK</td>
<td>few</td>
<td>few</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>FixK</td>
<td>few</td>
<td>many</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>SocialK</td>
<td>few</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

Gathering “it”: a.k.a. acquire

Making “it” run: a.k.a. operationalise

Finding bugs in “it”: a.k.a. fault (or realizing an enhancement is required).

Fixing “it’s” errors (or implementing the enhancement)

Ensuring that new fixes don’t damage old fixes: a.k.a. preserve

Rapid Acquire Assumption

1997 survey of readers of the comp.ai news group.

Q: What is your maintenance technique?

A: Rapid acquire systems (RAS):

- high-level environment
- maybe point-and-click graphical editors
- maybe operationalisation support.

The RAS assumption:

- If knowledge is expressed at a sufficiently high-level...
- then its flaws are obvious and quick to change.
- Yeah, right...
**RAS: Valid?**

Merely browsing knowledge may not reveal how the inference engine will use it at runtime.

Merely watching a program execute may not reveal subtleties in its behaviour (the operationalisation KM assumption, page 33).

Very short descriptions of knowledge may contain invisible faults (examples below).

---

**Small Models with Few Errors**

Population growth:

- [Levins and Puccia, 1985]:
  \[
  \frac{dN}{dT} = rN
  \]
  \(T = \text{time}, N = \text{population}, r > 0, r < 0 \text{ if environment benign, hostile respectively.}\)

- Error #1: \(\frac{dN}{dT} = 0\) when \(N \leq C\), the maximum carrying capacity of the environment

- Population growth (again):
  \[
  \frac{dN}{dT} = rN \left(1 - \frac{N}{C}\right)
  \]

- Error #2: Over-population \((N > C)\), hostile environment \(r < 0\), population increases (huh?)

Myers [Myers, 1977]: controlled experiments with 63 lines of PL/1

- 59 experienced data processing professionals
- Unlimited time
- 5 of the 15 errors found

---

**Ptype: Acquire**

Manual, unstructured, or semi-structured approaches.

Most SE/KE methodologies offer some of the following:

- A notation; e.g. ER [Date, 1995], Harel state charts [Harel, 1995], UML [Booch et al., 1997].

- Tool support drawing with (sometimes) operationalisation; e.g. RATIONAL-ROSE [Corporation, 1997], CAKE [Rich and Feldman, 1992].

- Representation of different stakeholders’ opinions, goals (e.g. [Ollé et al., 1991]) and combination rules (e.g. [Easterbrook and Nuseibeh, 1996]).

- Hints and tips:
  - On what data structures to collect; e.g. [Wielinga et al., 1992a, Ollé et al., 1991, Gamma et al., 1995, Buschmann et al., 1996, Fowler, 1997, Coad et al., 1997]
  - On software process; e.g. [Booch, 1996].

---

**Ptype: Acquire (2)**

Manual or semi-automatic, structured techniques, e.g. database normalization [Date, 1995] or knowledge normalization [Debenham, 1995]. Conceptual model: items connected by objects to describe:

- Data are simple variables.
- Information are relations connecting variables.
- Knowledge that execute over the relations.

**Decomposable item:** may be constructed from other items or objects. Knowledge normalization = discard decomposable data, information, knowledge [Debenham, 1995].

```
+--[part/sale-price, part/cost-price, mark-up]
    +--[mark up-rule]
    |   +---[part/cost-price]
    |   +---[cost price]
    +---[part/cost-price]
```

---
Acquire Using PSM Knowledge

Use formal methods to map from informal to formal knowledge [van Harmelen and Aben, 1996].

Restrict questioning to just those issues required to

- Select between PSMs
- Define PSM's ontological commitments (required data types)
- Refine selected PSM; e.g., if PSM = diagnosis, then ask about permissible obs in this domain.

\[
\text{data} \rightarrow \text{abstract} \rightarrow \text{obs} \rightarrow \text{hypothesis} \rightarrow \text{hyp}
\]

abstraction rules
causal rules


Some offer open-ended PSM definitions: PROTEGE-II, ?SBF
Others are harder to customize for a new PSM; e.g., SALT
Debatable point: are single-PSM tools (e.g., RIME based on SOAR, page 36) more or less flexible?

Ptype: Acquire for Specialised KTypes


VIEWER [Easterbrook and Nuseibeh, 1996] allows for the expression of inconsistencyK.

HT4 [Menzies and Compton, 1997]: tables for storing behaviouralK from experiments in the literature.

Case-based reasoning: builds behaviouralK via the incremental caching of parts of previous inferences (page 30).

Test generation tools: explore dependencies looking for inputs that exercise (e.g.) all branches: [Ginsberg, 1990, Zlatareva, 1992].

RDR: behaviouralK via incremental capture. In context of specific error, collect new knowledge. Remember the case that caused that error [Compton and Jansen, 1990].

BRAHMS: storage and organization of such socialK [Clancey et al., 1996].

Other Acquire Techniques

Index on post-conditions [van Harmelen and Aben, 1996]

Expert critiquing systems:

\[
\ldots \text{programs that first cause their user to maximize the falsifiability of their statements and then proceed to check to see if errors exist. A good critic program doubts and traps its user into revealing his or her errors. It then attempts to help the user make the necessary repairs [Silverman, 1992].}
\]

Critics offer cues to the user:

- Coaxes the user from useless issues they are considering to useful issues that they are ignoring.

Two types or critics:

- Influencers cue: prevents an error happening.
- Debiaser cues: fixes after errors have happened (see the Fix activity, below). Debiasers less useful without positive feedback (influencers) [Silverman and Wenig, 1993].

Ptype: Operationalisation

“Operationalisation”: the process of executing a KB either by direct interpretation or via compilation to some internal form.

E.g. SBF, PROTEGE-II, ..., TINA [Benjamins, 1994]. TINA KB:

\[
\text{rule1:diagnosis}
\text{when prime_diagnostic_method then symptom_detection and hypothesis_generation and hypothesis_discrimination.}
\]

\[
\text{rule2:symptom_detection}
\text{when ask_user_method then apply_user_judgment.}
\]

\[
\text{rule3:symptom_detection}
\text{when compare_symptom or detection_method then generate_expectation and compare.}
\]

\[
\text{rule4:hypothesis_generation}
\text{when empirical_hypothesis_generation_method then associate and prediction_filter.}
\]

\[
\text{rule5:hypothesis_generation}
\text{when model_based_hypothesis_method then find_contributors and transform_to_hypothesis_set and prediction_based_filtering.}
\]

\[
\text{rule6:hypothesis_generation}
\text{when hypothesis_generation_method then select_hypothesis and collect_data and interpret_data.}
\]
Ptype: Operationalisation (2)

TINA output (after some queries to the user’s db regarding the rules on the previous page).

```plaintext
model_based_hypothesis_generation_method {
  trace_back_method;
  intersection_method;
  corroboration
}

trace_back_method {
  find_upstream
}

intersection_method {
  intersection
}

corroboration_method {
  select_random;
  simulate_hypothesis;
  compare;
  delete
}
```

Ptype: Fault

*Fault*: recognizing that an operationalized KB has produced a behaviour that needs changing (bugs or enhancements).

Fault techniques for wordK: see repertory grids, page 45.

Fault techniques for sentenceK and PSMs:

- Expert inspection: use knowledge not captured in the 7 Ktypes
- BehaviourK
- Using QualityK, inconsistencyK
  - Heuristic knowledge, may need maintenance.
  - No research known into how to fault qualityK, inconsistencyK (the recursive maintenance problem, see page 48).

Fault techniques for behaviouralK:

1. (Via the recursive maintenance problem): use some KB2 to model expectations of values in KB1’s behaviouralK.
2. Use some algorithm to detect if new data does not fit into old data; e.g. [Colomb, 1999].
3. Via coverage of some feature X. Demand extensions to behaviourK if coverage inadequate?

Ptype: Fault, subtype: Browse-around

Let the user manually generate their own explanations of why a variable was/was not set via:

- Fault localization (see below).
- MYCIN’s “how” and “why” queries.
  - How= path to this point.
  - Why= current goal of backward chaining.
- SALT’s “how” and “why not” and “what if”.
  - “Why not X”= given conclusion Y, find a path to X blocked by some contribution to Y.
  - “What If X”= A hypothetical look downstream of some temporary setting. Variables set in what-if mode must be reset. HT4 a better approach (page 31).
  - No “why” since SALT was a forward-chainer.


- KIV (an interactive theorem prover) cannot solve a problem using a PSM.
- Identify, automatically, the missing logical formula that blocks PSM completion.
- Offer same to user as an assumption A.
- Let a user browse-around a first-order theory representing the PSMs in a “what if A”?
**Ptype: Fault, subtype: Fault localization**

Via backwards search of word dependencies:

- Used in the MYCIN rule editor [Davis, 1976], Darden's anomaly localization [Darden, 1990], model-based diagnosis [Hamscher et al., 1992], RDR [Compton and Jansen, 1990],...

- Important technique


- Patch tree of rules
- Each rule has an unless patch (which is another rule and so on recursively).
- Patches stored with case that prompted patch creation.
- Final conclusion = conclusion of the last satisfied rule.
- Fault = the path over the patch tree to that rule.

Note: little or no recursive maintenance problem in RDR.

---

**Ptype: Fix**

“Fix”: the process of removing a fault in an operationalized KB.

- Special class of “fix” = conflict resolution for knowledge collected from different sources.

Some general mechanisms for fixing:

- Ripple-down rules
- Conflict negotiation
- Specialization and/or generalization
- Machine learning
- Case-based reasoning
- KA scripts
- Others as described in [Menzies, 1999].

---

**An RDR Tree**

```
1 a & b x1
else unless

2 c x2
else unless

3 d x3
```

**Fix via Ripple-down-rules**

RDR = fault localization + repair:

- Current case = C = Feature1, Feature2, ...
- Path to faulty rule = P = Feature3, Feature4, ...
- Candidates for repair = difference list = C - P.
- Show difference list to an oracle of repair. Oracle picks X items off the difference list. These are added to new patch.

Patch only in the context of the last error? Crazy! Crazy?

- PIERS: St. Vincent’s Hospital, Sydney,
- Modeled 20% of human biochemistry sufficiently well to make diagnoses that are 99% accurate [Preston et al., 1993].
- Development blended seamlessly with maintenance
- 2000-rule system, maintenance = a few minutes each day by domain experts.
RDR: Pros, Cons

Advantages:

- Practical implementation technique for the reflective [Schon, 1983] or situated cognition view [Menzies and Clancey, 1998]:
  - Design mostly happens when some concrete artifact talks back to the designer—typically by failing in some important situation.
  - Less concerned with the creation of some initial artifact than the on-going re-interpretation and adjustment of that artifact.
- Supports “preserve”: new fixes don’t break old fixes.
- Mostly avoids the recursive maintenance problem.
- Contrast with [de Brug et al., 1986]: large expert systems are notoriously hard to maintain.

Disadvantages:

- RDR tree not compatible with other common representation types e.g. isa hierarchies, state charts)- but see [Richards and Compton, 1997, Lee and Compton, 1996, Colomb, 1999].
- Can’t process meta-knowledge such as PSMs - but see [Menzies and Mahidadia, 1997].

Fix via Negotiation

Explicitly represent the different viewpoints of different users.

Automatically detect conflicts between these viewpoints.

- Each inconsistency detection rule has an associated repair action.

Offer tools for resolution support.

Side effect of resolution = knowledge refinement [Easterbrook and Nuseibeh, 1996].

Note: conflicts may not be resolvable now.

- Need to be able to reason onwards in the presence of inconsistency.

Fixing via Case-Based Reasoning

Recall our heart attack patient, page 30 [Kolodner, 1993, p419]

New case: Dashed= added edges. Deleted: “murmur of as”.

CBR: Pros/Cons

CBR: strong on fixing cases (behaviouralK); weak on maintaining:

- Maintaining fixK: algorithms for turning old into new;
- SentenceK: the background causal knowledge or cardiac behaviour used above.
Fixing via specialization/ generalization

[Shapiro, 1983]. Undesired behaviours can be removed by specializing a pre-condition;
- e.g. increasing the number of tests in a conjunction.

Desired behaviour which was not achieved can be reached via generalizing a pre-condition;
- e.g. decreasing the number of tests in a conjunction.

Human-in-the-loop spec/gen:
- RDR: start with ultimate generalization: true. Specialize with each patch.
- SEEK: specialized by adding tests/symptoms or deleting exclusions or decreasing its confidence level. Generalized by removing tests/symptoms or adding exclusions or increasing its confidence level.

Fully automatic spec/gen:
- SEEK2 (page 46).
- Machine learning

Fix via Machine Learning

Machine learning (ML) algorithms [Michalski, 1993]:
- Input:
  - BehaviouralK: positive and negative examples ($E^+, E^-$) of what you want, don’t want.
  - Background theory ($B$) which may be empty
- Output:
  - A new theory which covers more of $E^+$ or less of $E^-$ than the initial background theory.
- Two types: inductive and deductive.

Inductive Learners

E.G. genetic algorithms, neural nets, decision tree learners, belief networks

Create a summary theory from the behaviouralK presented to them.

Data hungry: best with large $E$ (100s to 1,000s)

Problems with very large $E$ (1,000s): need pre-processors to prep the data (the data mining problem).

Typically ignore background knowledge (exception: inductive logic programming [Muggleton, 1991]):
- A user may be presented with a totally novel theory at the end of an inductive learning session.
  - BAD IDEA?
  - Users treasure their favorite portions of their KB (typically, the bits they defended from all critics).
  - Learners should not scribble all over treasured knowledge.

Deductive Learners

E.G. explanation-based generalization [van Harmelen and Bundy, 1988, Mitchell et al., 1986], chunking in rule-based systems [Laird et al., 1986]

Deductive learners input the steps taken by some inference engine and output a better set of inference steps.
- May cull the middle portions of a long inference procedure to connect inputs directly to outputs.

No scribble problem.

Less data hungry and make extensive use of the background theory.
- BAD IDEA?
- Less information to learn from = ? local minima.
**Machine Learning and KM**

Deductive learning less reliable than inductive (the local minima problem).

Inductive learners have practical drawbacks:

- The scribble problem (see above).
- Data hungry = more expensive. May take months to collect data.

Manual KA using RDR < inductive learning when $E > 1000s$ [Mansuri et al., 1991]

---

**Fixing via KA Scripts**

Databases: transaction management ensures completion of all table updates/deletes.

Knowledge bases: EXPECT Transaction Manager (ETM) ensures completion of all PSM knowledge updates/deletes:

- Triggered when EXPECT’s partial evaluation strategy detects a fault.

<table>
<thead>
<tr>
<th>Simple task #1</th>
<th>Harder task #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>no ETM</td>
</tr>
<tr>
<td>**</td>
<td>S4</td>
</tr>
<tr>
<td>Total time (min)</td>
<td>25</td>
</tr>
<tr>
<td>Time completing transactions</td>
<td>16</td>
</tr>
<tr>
<td>Total changes</td>
<td>3</td>
</tr>
<tr>
<td>Changes made automatically</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Recursive maintenance problem: how to check the KA scripts?

---

**Ptype: Preserve**

Stop a change for issue D from introducing problems into changes to made for A,B,C, . . .

Logic-based preserve:

- Initially, structure to support preserve.
- E.G. RDR (page 64), database and knowledge normalization (page 55).

Network-based preserve

- At all times, have access to the dependency network to check implications of a change.

Procedural-based preserve:

- Use a transaction manager to ensure completions
- E.G. KA scripts, page 77.

---

**Other Preserve Tools**

Design rationale [Moran and Carroll, 1996]:

- An annotation that describes the reasons for a fix.
- Current state-of-the-art:
  - Annotation tools for wordK and sentenceK only.
  - Creating such argumentation structures very costly to build. Interesting approach: Use CBR as a secretary to compare enrich new arguments by reflecting over old arguments [Fischer et al., 1996]

Data schema evolution (DSE).

- After some change to the logical model of the program, some parts of the program comply to the former version of the schema.
- Pressing problem in OODBMS (complex structures).
- Future problem for OO KBs (ontologies)
- Chain different schema versions, class to class.
- Coercion functions map instances along the versions.
- Current commercial state-of-the-art: can’t handle changes that transcend class boundaries [Odberg, 1995, chpt2].
Discussion: How to Commission a KM Tool

<table>
<thead>
<tr>
<th>Preliminaries</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial:</td>
<td>6</td>
</tr>
<tr>
<td>Objective</td>
<td></td>
</tr>
<tr>
<td>Brief Description</td>
<td></td>
</tr>
<tr>
<td>Audience</td>
<td></td>
</tr>
<tr>
<td>Expectation management</td>
<td></td>
</tr>
<tr>
<td>About the author</td>
<td></td>
</tr>
<tr>
<td>Author biases</td>
<td></td>
</tr>
<tr>
<td>The KM space</td>
<td>25</td>
</tr>
<tr>
<td>5 processing activities</td>
<td></td>
</tr>
<tr>
<td>Discussion</td>
<td>80</td>
</tr>
<tr>
<td>How to commission a new KM tool.</td>
<td></td>
</tr>
<tr>
<td>References</td>
<td>82</td>
</tr>
</tbody>
</table>

Search-Space Reflection

“Browse-around” core technique for KM. Used everywhere:

- Normalization = browse the dependency network
- Fault-localization
- “How”, “why”, “why not” and “what if” = browse dependencies around some word.
- Deductive learners = browse and edit pathways leading to a conclusion.
- CBR = browse which bits of the KB were used before.
- HT4 = browse around and sort out what can be believed together.
- Requirements engineering = browse around to find fixes to stake-holder conflicts. ...

Does the tool support browse-around?

4 Issues

1. Where in the 5*7 points of KM space does this system work?
2. What tools are offered for maintaining the maintenance KB? (the recursive maintenance problem).
3. Theoretically: Does the tool support preserve?
4. Practically, check that:
   - QualityK can assess the KB;
   - We can track \( \frac{dQ}{dT} \) (quality): very useful for project management.

References

For the tutorial: Knowledge Maintenance: The State of the Art by Tim Menzies tim@menzies.com


