

hQkb- The High Quality Knowledge Base Initiative (Sisyphus V: Learning Design Assessment Knowledge)

Tim Menzies

NASA/WVU IV&V Facility, 100 University Drive, Fairmont WV 26554

<tim@menzies.com>

Abstract

Previous attempts to evaluate different KEng techniques have labored under at least four difficulties: (1) lack of objective measures of success; (2) lack of success measures that are meaningful to the KBS user community; (3) lack of baseline values to make measurements meaningful; and (4) certain restrictions in the range of technologies assessed. In this paper, an accessible theory of knowledge engineering is presented: a good KEng-product offers high quality support for a design&learn loop. The high quality knowledge base initiative (hQkb) operationalizes that theory, plus some objective measures of system quality. The hQkb evaluation will be focused on external quality attributes (e.g. predictability, survivability, adaptability) of a wide range of techniques. To ensure objectivity, these quality attributes would be evaluated using specially hired verification and validation consultants.

1 Introduction

The goal of much of automated software and knowledge engineering is the optimization of code generation. For example, one of the goals of NASA's *Intelligent Systems initiative*¹ is the creation of tools that can generate millions of lines of codes in tens of seconds. To achieve this goal, current best practice must be identified and automated. But what shall we automate? To put this another way, what represents current best practice in software and knowledge engineering (hereafter SEng and KEng)? Where can we look to find descriptions of demonstrably best practice?

One place we can look should be comparative evaluation experiments. Several exist in the KEng literature:

The Oak Ridge Study: Oak Ridge set a challenge problem (inland oil and hazardous chemical spills) and had a group of knowledge engineering researchers implement solutions [Barstow, Aiello, Duda, Erman, Forgy, Gorlin, Greiner, Lenat, London, McDermott, Nii, Politakis, Reboh, Rosenchein, Scott, van Melle & Weiss 1983].

DARPA's high performance knowledge base initiative (HPKB). HPKB aims at improving the rates we can build KBs by one to two orders of magnitude. For notes on HPKB, see Figure 3.

The Sisyphus project of the Banff community: Sisyphus is a range of projects, three of which took the form of the Oak Ridge study. For notes on Sisyphus, see Figure 4.



Figure 1: The hQkb logo: any software system should clearly record its performance.

¹<http://actuva-www.larc.nasa.gov/techplan/4.0/#4.2>

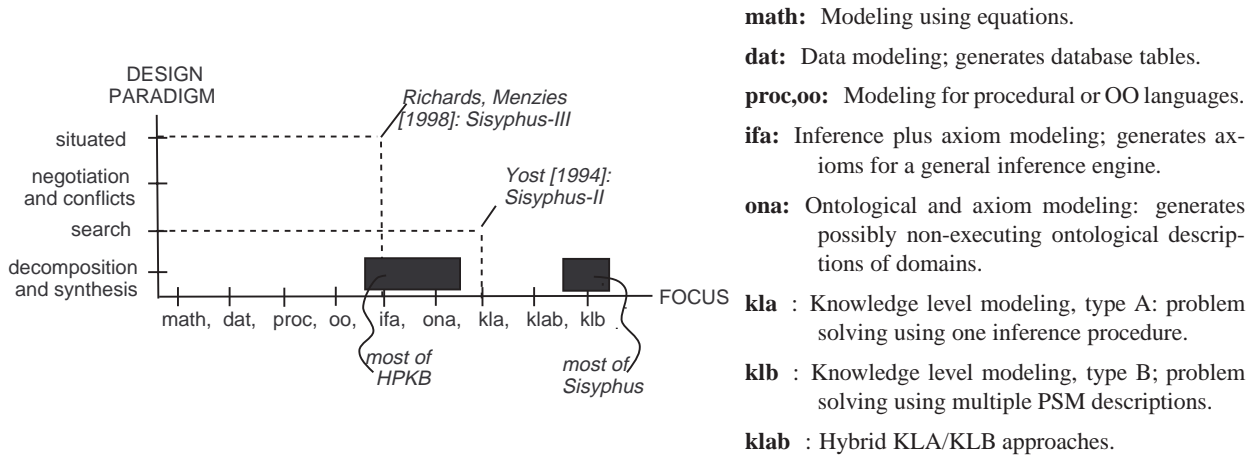


Figure 2: A KEng-methodology is a combination of design paradigm, focus, (shown here) plus tools and resources (not shown). Focus types are defined on the right-hand-side and discussed further around Figure 6. For more information on design paradigms, see Figure 5. The standard HPKB or Sisyphus project sampled only a small subset of the design paradigm vs focus space (exceptions: [Richards & Menzies 1998, Yost 1994]).

I argue that none of these evaluations let us find best practice. Consider the case of a NASA software manager asking the *launch question*; i.e. “can I entrust my satellite to a knowledge base system (KBS)?”. Nowhere in the Oak Ridge, HPKB, or Sisyphus evaluation results is a *quality statement* of the form “we can certify that this KBS will work, given the known operational conditions seen in the mission profile”. More generally, the evaluation criteria used in the Oak Ridge, HPKB, and Sisyphus studies address the concerns of the KBS developer community. *These studies do not address the concerns of the KBS user communities.* For example, much of Sisyphus work is based around a technology called *problem solving methods* (PSMs) which we define below. One conclusion from the Sisyphus studies is that PSMs are a rich engineering framework for discussing and implementing KBS. Note that this conclusion deals solely with the concerns of the system development community. This conclusion says nothing about the reliability or quality of the generated systems. Such quality statements are required if we are to answer user concerns such as the launch question.

The aim of the high quality knowledge base initiative (hQkb) is to address this lack of quality-based evaluation results. In hQkb, an international consortium of knowledge engineering (KEng) researchers would take some specification and use different techniques to develop a working system. The different products would then be comparatively assessed based on external quality attributes; e.g. system maintainability, adaptability, reusability, testability, controlability, etc. The “Q” in hQkb is capitalized to remind us that the stand-out feature of hQkb is a focus on quality measures. hQkb will use three methods for this quality assessment:

Wide range of systems: One method of assessing a methodology M is to:

1. Clearly distinguish it from a competing methodology $\neg M$.
2. Comparatively assess M by measuring variables collected from both M and $\neg M$.

Much of HPKB and Sisyphus comes from only a small region in the space of known approaches; see Figure 2. That is, HPKB and Sisyphus did not define $\neg M$ nor collected comparative results from a $\neg M$ approach.

Process-based assessment: All hQkb implementations will instantiate the same software process (the Design&Learn loop described below). We will say that the higher the quality of an hQkb offering, the

Goal: To foster the development of technologies that can increase the rate at which we can write knowledge bases.

Baseline: Current KB authoring rates average at 5 axioms per hour, 10,000 axioms per year.

Aim: To increase the baseline by one to two orders of magnitude.

Organization: DARPA funds bi-annual meetings and two “intergration teams” (SAIC and Teknowledge) whose role is to build unified workbenches from the contributions of HPKB participants.

Participants: HPKB participants come, for the most part, from the United States. These participants, and the principle investigators, include *CYCORP*:Lenat; *Stanford*: Fikes, Koller, McCarthy, Wiederhold, Musen; *George Mason University*: Tecuci; *Carnegie Mellon University*: Mitchell; *University of Massachusetts*: Cohen; *ISI*: MacGregor, Patil, Swartout, Gil; *SRI*: Lowrance, desJardins, Goldszmidt; *Kestrel*: Espinosa; *MIT*: Doyle, Katz; *Northwestern University*: Forbus; *AIAI at Uni. Edinburgh*: Kingston, Tate; *TextWise*: Liddy, Hendler.

Biases: HPKB ignored, for the most part, the problem solving methods (PSMs) research (see the discussion below in the section: *The Range of Treatments*). PSMs are a major focus of the Sisyphus project(Figure 4).

Results:

1. In HPKB year one, the George Mason team generated the most new axioms added per day (787 binary predicates) using DISCIPLINE: an incremental knowledge acquisition tool [Tecuci 1998]. DISCIPLINE includes machine learning tools for abstracting learnt rules which makes them more generally applicable. As DISCIPLINE runs, it builds and updates the meta-knowledge used for the purposes of abstraction.
2. Cohen, Chaudhri, Pease & Schrag [1999] studied how much ontologies supported the development of HPKB applications. The recent terms added to an ontology offer more support than words added previously by other authors. My reading of this result is that it does not support the current efforts in building supposedly reusable ontologies.

For more information: See <http://www.teknowledge.com/HPKB/> and [Cohen, Schrag, Jones, Pease, Lin, Starr, Gunning & Burke 1998].

Figure 3: DARPA’s high-performance KB (HPKB) initiative: notes

more support it brings to Design&Learn.

Centralized independent assessment: Developers would not assess system quality. Repeated empirical studies show that the developer of program *X* is the worst audience to assess that program *X* [Jones 1998, p558]. All hQkb products will be assessed at NASA’s independent verification and validation facility².

Search-based assessment : I have been developing a theory of search-based agents for automatically assessing KBS quality. Part of the hQkb project is an exploration of this theory.

The rest of this article explores the following three facets of hQkb, in the order listed below. In any experiment, a *range of treatments* are applied to a *system* and some *effect* is measured:

The range of treatments: hQkb will encourage developers to implement a system using a wide range of techniques.

The system: Design&Learn is my generalization of Josephson, Chandrasekaran, Carroll, Iyer, Wasacz & Rizzoni’s [1998] abstract model of design. Design&Learn will be the meta-framework within which the hQkb members will work.

The measured effects: The different Design&Learn implementations will be assessed via their external quality attributes.

²Note: funding to hire this evaluation team is being applied for and, if approved, would not be available till mid-2000, at the earliest.

The Sisyphus projects are a series of challenge problems in which a knowledge acquisition problem was defined and tool developers were challenged to solve it with their tools. Unlike HPKB, Sisyphus was run by a loose consortium of international KEng researchers on a shoestring budget. Sisyphus began in 1990 and continues to this day. Communication is mostly via email and status reports at the semi-annual Banff KA workshops. Sisyphus is mostly populated via Europeans but some cross-over with the HPKB community exists (i.e. Musen, Gil).

A significant bias in the Sisyphus projects was towards problem solving methods (PSMs) research (exceptions: [Richards & Menzies 1998, Yost 1994]). In this regard, Sisyphus is very different to HPKB (Figure 3).

There have been four Sisyphus projects defined:

Sisyphus-I: Room Allocation A number of people with differing requirements have to be allocated appropriate rooms with differing characteristics. It was based on an ESPRIT study at the GMD in Germany and was aimed at KEng tasks not focused heuristic classification [Linster 1992, Clancey 1985].

Sisyphus-II: Elevator Configuration The room allocation problem proved to be relatively simple and the second Sisyphus project attempted to provide a realistic knowledge-engineering problem based on the VT system [Schreiber & Birmingham 1996, Marcus, Stout & McDermott 1987].

Sisyphus-III: Lunar Igneous Rock Classification The principal objectives of the Sisyphus III project were:

- To provide for better quantitative comparison of KB systems and the methodologies employed to build them, through use of a set of achievement metrics
- To provide more realistic access to actual KA material in a staged series of releases
- To obtain more complete records (or knowledge engineering meta-protocols) concerning the processes that the knowledge engineer goes through in the KBS construction process [Shadbolt, O'Hara & Crow 2000].

Sisyphus-IV: Integration over the Web Sisyphus-IV is a project to encourage collaborative use of tools through the net and web and by the integration of web tools at different sites through the net.

While Sisyphus has unified a diverse range of researchers, and hundreds of publications have been generated, the objective evaluation results to date are inconclusive:

... none of the Sisyphus experiments have yielded much evaluation information (though at the time of this writing Sisyphus-III is not yet complete) [Shadbolt et al. 2000].

Nevertheless, the Sisyphus researchers remain optimistic and the project continues.

Figure 4: Notes on the Sisyphus initiatives. Extended from notes at <http://ksi.cpsc.ucalgary.ca/KAW/Sisyphus/>.

2 The Range of Treatments

hQkb's treatments will be the different methodologies used to create some KEng-product. This section describes those methodologies. The methodologies surveyed by hQkb will be very wide. There are two motivations for this:

Technology biases in existing studies: Figure 2 claimed that HPKB and Sisyphus only sampled a small portion of the space of KEng methodologies. For more notes on that claim, see the end of this section.

Baselines: Recalling the introduction, it is very useful to calibrate measurements from sophisticated techniques with measurements from not-so-sophisticated techniques (but be aware that sometimes, the "straw-man" will not burn).

Software construction techniques can be characterized by the process used to create it; the interim and final products; and the resources used to create it [Fenton & Pfleeger 1997, chpt3]. A detailed characterization of software process, product, and resources is still an open research issue. hQkb will focus on the parts of process-product-resources which can be reasonably well defined; i.e. tool, resources, design paradigm, and focus.

Resources and tool: hQkb members would have to describe the tools used to develop their system, the time taken, and make some indication about the skill of the developer(s).

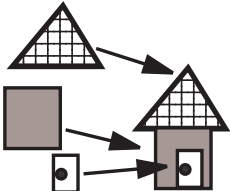
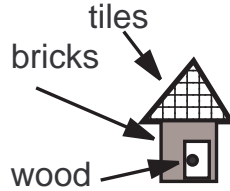
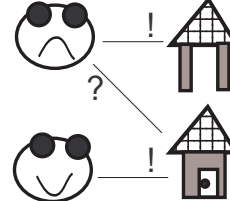
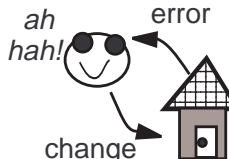
<i>Design paradigm:</i>	Decomposition and synthesis	Search	Negotiation and deliberation	Situated
<i>Driver:</i>	Reuse libraries	Heuristics	Conflicts	Errors
<i>Early work:</i>	[Alexander 1964, Alexander, Ishikawa, Silverstein, Jacobsen, Fiksdahl-King & Angel 1977]	[Simon 1969]	[Webber 1973, Finkelstein, Gabbay, Hunter, Kramer & Nuseibeh 1994]	[Schon 1983, Shalin, Geddes, Bertram, Szczepkowski & Dubois 1997]
<i>Current work:</i>	OO design patterns [Menzies 1997], KADS [Wielinga, Schreiber & Breuker 1992]	SOAR [Rosenbloom, Laird & Newell 1993], HT4 [Menzies 1998b], SVF [Josephson et al. 1998]	Viewpoints in requirements engineering [Finkelstein et al. 1994]	RDR [Compton & Jansen 1990, Preston, Edwards & Compton 1993]
<i>Example:</i>	Build the house using parts found when the last house was built. 	Build the house via a search of the space of what is known of bricks, tiles, and wood. 	Let each stakeholder design their preferred house, then lets discuss it. 	Build the 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. 

Figure 5: Building a house using four different paradigms of design

Design paradigm: In general, process is hard to characterize. How do we know (e.g.) if a developer says they use Booch's [1996] OO development method, that they are really following that recipe? However, Moran & Carroll [1996, p3] argue that four broad descriptions of design paradigms exist in the literature (Figure 5). Notes on these paradigms follow:

Design as decomposition and synthesis: A proponent of design as decomposition and synthesis might predict that using well-developed software libraries will mean fewer loops of Design&Learn than working everything out from scratch. This paradigm dates back at least to 1964 with Alexander's work on architecture [Alexander 1964, Alexander et al. 1977]. Design is taken to be the re-shuffling of components developed previously, then abstracted into a reusable form. Modern expressions of this approach include object-oriented design patterns [Buschmann, Meunier, Rohnert, Sommerlad & Stal 1996, Menzies 1997], and the knowledge engineering research into ontologies [Gruber 1993, van Heust, Schreiber & Wielinga 1997] and problem solving methods [Chandrasekaran 1983, Clancey 1985, Schreiber, Wielinga, Akkermans, Velde & de Hoog 1994]. Our reading of the literature is that this approach is the dominant paradigm in contemporary knowledge and software engineering. Elsewhere, I have discussed limitations to this paradigm [Menzies 1998b, Menzies 1998a].

Design as search: A proponent of design as search might predict that structuring Design&Learn around search primitives will result in an adaptable Design&Learn system faster than other approaches. This paradigm dates back at least to 1969 with Simon's work on artificial intelligence [Simon 1969]. Design is taken to be the traversal of a space of possibilities, looking for pathways to goals. Modern expressions of this approach includes most of the AI search literature [Pearl & Korf 1987] and the SOAR

knowledge-level (KL) search [Newell 1982, Newell 1993]. In a KL search, intelligence is modeled 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. Note that the problem solving methods/ontology community also calls its approach “knowledge-level modeling”. However, that community and Newell implemented very different architectures. We will call Newell-style knowledge-level modeling KLA and the other kind of knowledge-level modeling KLB.

Design as deliberation and negotiation: A proponent of design as deliberation and negotiation might predict that the better the Design&Learn debate module, the better the Design&Learn system. This paradigm dates back to at least 1972 with Rittel’s work on *wicked problems* [Rittel 1972]. 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. Modern 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 Easterbrook [1991], Finkelstein et al. [1994], and [Nuseibeh 1997] 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.

Situated Design: Proponents of situated design might predict that minimal modeling should be performed in Design&Learn prior to running specific examples. This paradigm dates back to at least 1983 with Schon’s work on the *reflective practioner* [Schon 1983]. In this approach, design mostly happens when some concrete artifact *talks back* to the designer- typically by failing in some important situation. That is, reflective design is less concerned with the creation of some initial artifact than the on-going re-interpretation and adjustment of that artifact. Modern expressions of this approach include the situated cognition community [Suchman 1993, Clancey 1989], certain approaches to design rationale [Fischer, Lemke, McCall & Morch 1996, Casady 1996], and knowledge engineering techniques that focus on maintenance rather than initial design [Compton & Jansen 1990, Menzies 1998b, Menzies & Clancey 1998]. In the design as reflection paradigm, testing for model failure is an essential, on-going process.

Another paradigm: The design paradigms surveyed by Moran & Carroll do not explicitly mention a fifth design paradigm: the development of new components. However, we could infer that the generation of new components arises out of design as search, design as deliberation and negotiation, and situated design.

Focus: After resources, tool, and design paradigm, the other hQkb treatment characterization is the *primary modeling focus*. Figure 6 shows this dimension of KEng-methodologies. Each point in this dimension (MATH, DAT, PROC, OO, IFA, ONA, KLA, KLAB, KLB) uses some different combination of primary modeling constructs, as described in Appendix A:

MATH: *MATHeMATical modeling*. Impressive and intuitive visual programming environments exist that allow end-users to model complex systems, then execute them using some equation solver [Inc. 1994, Gautier & Gruber 1993]. One advantage of the MATHs-based approach is that certain quality factors (stability, observability, controlability) can be rigorously determined [Ishida 1989].

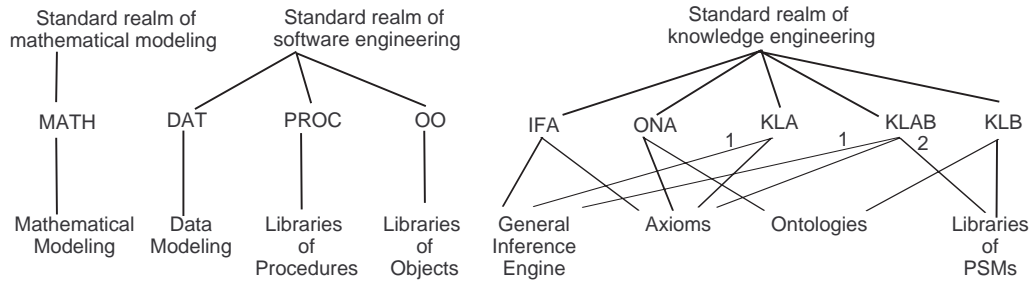


Figure 6: Soft construction techniques can be divided on what combinations of primary modeling construct they use. The terms in the last row are defined in Appendix A. “1” denotes that the single inference engine is customizable; e.g. the knowledge engineer can provide operator selection rules to customize the problem space traversal [Laird & Newell 1983]. “2” denotes that PSMs in KLAB are used only in an initial analysis stage.

DAT: *DATA* modeling. The data modeling group who target relational databases. Theoretically, there is nothing stopping data modeling workers from developing KBS [Ullman 1988]. However, in practice, conventional database manipulation languages are much stronger on IO functions and disc storage than intricate RAM-based manipulation of data.

PROC, OO: *PROC*edural, *Object Oriented* modeling. Rejects the declarative representations used in the other approaches. In the 70s, this was a large research area. Proponents of frame representations (e.g. [Winograd 1975, Minsky 1975]) 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 [Nilsson 1991, Birnbaum 1991] but the complexity of reasoning about procedures (e.g. [Etherington & Reiter 1983]) drove most researchers to declarative characterizations of their frame-based knowledge (e.g. [Brachman, Gilbert & Levesque 1989]). Pure PROC/OO KEng researchers are rare these days, but some still keep the faith (e.g. [Birnbaum 1991, Brooks 1991]). Few PROC/OO researchers are known amongst the HPKB and Sisyphus communities.

IFA: *InF*erence + *Axioms*. No explicit representation meta-knowledge when modeling KBS and a single inference procedure. Crudely expressed, in the IFA approach, KEng is just a matter of stuffing axioms into an inference engine and letting the inference engine work it all out. Successful IFA variants provide rigid control on how new axioms are asserted; e.g. [Compton & Jansen 1990].

ONA: *ON*tologies + *Axioms*. Active focus on ontology creation. 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. [Gamma, Helm, Johnson & Vlissides 1995]).

KLA: *Knowledge-Level modeling type B- Newell-style*. A strong commitment to a single inference procedure, which can be customized. This inference procedure features predominantly when modeling a system; e.g. [Yost & Newell 1989, Laird & Newell 1983, Menzies & Mahidadia 1997]. To the best of my knowledge, all the known KLA developers use design-as-search.

KLAB: 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 [Chandrasekaran, Johnson & Smith 1992].

KLB: *Knowledge-Level modeling type A- non-Newell style.* Catalogues libraries of PSMs [Wielinga et al. 1992] or explores a single PSM within such a library [Marcus & McDermott 1989]. Extensive use of ontologies. At runtime, KLB may use a general inference engine to execute their systems (e.g. older versions of PROTEGE-II [Eriksson, Shahar, Tu, Puerta & Musen 1995] compiled down to CLIPS [NASA 1991]) but this inference engine does not feature in the design discussions.

While generalizations are hard to form from the diverse international KEng research community, it is roughly true that HPKB focused on IFA and ONA while Sisyphus focuses on KLB. Roughly speaking, in the HPKB view, PSMs are ignorable and axioms/ontologies are all; e.g. [Lenat & Gutha 1990]. One of the reasons for this difference in the two communities is geography. Most of the Sisyphus researchers came from Europe where the KADS-PSM project [Wielinga et al. 1992] was very influential. On the other hand, most of the HPKB researchers came from the United States where the CYC project has a high profile [Lenat & Gutha 1990]. The CYC project focuses on ontologies, not PSMs.

Curiously, even though this international division exists in the KEng community, comparative data on the merits of the different focuses are very few. These data points come from the Sisyphus project:

- In the Sisyphus-II initiative, Yost [1994] built a KLA elevator configuration system using design-as-search while the other teams used KLB [Schreiber & Birmingham 1996]. Yost's reported development times were much less than any of the other teams.
- In the Sisyphus-III initiative, Richards & Menzies [1998], used a IFA/situated approach (Ripple-down-rules) to produce a working system before the other KLB teams.

These two data points are hardly conclusive. For example, Yost had analysed the domain extensively prior to system construction. That is, it is possible that Yost's extra experience with the domain gave him an advantage. Nevertheless, the one general conclusion we would offer is that more data points should be collected. Hence hQkb.

3 The System

The previous section defined the hQkb treatments; i.e. the different ways we can build systems. This section addresses the next logical question: what specific system will be built?

This section describes **Design&Learn**: a meta-level description of system design and assessment. It is processed that hQkb participants structure their implementations around **Design&Learn**. We focus on **Design&Learn** for three reasons:

1. NASA may choose to fund hQkb, but only if it impacts on current NASA projects. Two such current projects are the Intelligent Synthesis Environment (ISE)³ and the Independent Verification and Validation (IV&V) facility⁴. ISE aims to support distributed design of NASA systems. IV&V seeks to find errors in NASA software. Hence, we focus hQkb on the creation and assessment of designs.
2. Quality assessment measures require some definition of the purpose of a system. **Design&Learn** offers one generic description of such a purpose: the development and evaluation of software designs.
3. The discussion at the end of this section will argue that **Design&Learn** is general enough to accommodate KEngs from different design paradigms.

³<http://actuva-www.larc.nasa.gov/techplan/4.0/#4.6>

⁴<http://www.ivv.nasa.gov/>

To define Design&Learn, we begin with the generic statement from Simon [1969]; i.e.

$$design = generate + test$$

Tacit in that characterization is at least following structures:

Ideas: Concepts that collectively create the space searched during design. For example, we might be working with a list of car parts.

Constraints: Knowledge of legal combinations of those ideas. For example, the marketing department might have told us to not make cars with red doors.

Assessment: Dimensions on which we can assess a design. For example, we may know that our customers care about how many miles per gallon our cars will use during city driving. Assessments may be simple dimensions or some combination of dimensions. For example, we might know that our customers would prefer a car that uses little gasoline over one that goes very fast.

Also tacit in Simon’s characterization are the following issues:

Debates: The arguments humans have around the design process. Such debates can mature the assessment criteria.

Iteration: Design is a looping process that repeats every time a designer says “I’ll just change this and see what I get”.

Pragmatics: Design is a resource bound activity which stops when the designers run out of time or find a satisfactory solution.

Learning: We would hope that lessons learnt in one pass of the design cycle are used to improve the next pass of the design cycle. That is, after each loop of an iterative design process, it should be possible to speed-up the next loop. The classic optimization technique is to move the test criteria into the generate process; i.e. don’t waste time generating what will be culled anyway by the test criteria. In this framework, that would mean changing the ideas, constraints, and assessment criteria.

Figure 7 shows how the above ideas interact. This characterization contains two extensions to the standard design-as-search paradigm (e.g. [Motta & Zdrahal 1998]). A large class of design systems leave the user out of the loop and assume static design assessment knowledge. In this characterization:

- The human-in-the-loop appears as the **debate** function.
- The loop allows for the evolution of design knowledge. Two artifacts are created: the best option design and better design knowledge in the form of matured assessment criteria. Such revised assessment criteria could be used for future applications.

```

procedure Design&Learn(
    ideas,      {e.g. list of car parts}
    constraints, {e.g. no red doors}
    assessment, {e.g. city mph, top speed, fuel economy
                more important than speed}
); best
1. while not outOfTime
2. do options ← test(generate(ideas, constraints))
3.   assessment ← debate(assessment, options)
4.   best ← top(rank(options, assessment))
5a.  if ok(best)
5b.  then return best
    else < ideas, constraints, assessment > ←
5c.  learn(assessment, options)
    fi
    done
6. return best
end Design&Learn

```

Figure 7: Design&Learn

An important premise in hQkb is that **Design&Learn** offers enough engineering options to support researchers from different design paradigms. To test this premise, we explore how the different paradigms would tackle the implementation issues of **Design&Learn**.

Designers using deliberation and negotiation might focus on the **debate** function. Their rationale might be that the key issue is encouraging community buy-in to the next version of the design.

Designers using decomposition and synthesis might implement **Design&Learn** via pre-defining a rich ontology of design and assessment. Such an ontology, they might argue, represents the total space of design and assessment options. Editors based on that ontology could support a wide range of **learning**. That is, decomposition and synthesis designers would focus on the initialization of the **ideas** passed into the initial loop of **Design&Learn**.

Alternatively, designers using situated design might focus on the **learn** function. In direct contrast to the above decomposition-and-synthesis design, situated designers might minimize the ontological commitments of the whole system in order to increase the **learn** flexibility.

Lastly, designers using search techniques might have another focus. They might argue that the key issue is covering the range of design options. Their focus might hence be in the **test** and **generate** part of **Design&Learn**.

4 The Measurement of Effects

Previously we have defined a system (**Design&Learn**) and some treatments for that system (constructing applications using different design paradigms and focus). When the treatment is applied, an hQkb member will use some KEng-tool to generate a KEng-product. In this section we discuss how we will assess those KEng-tools and KEng-products.

The assessment scheme is as follows:

1. KEng-products will all be built from the same specification. That specification must refer to some domain that is both broadly understood and of interest to potential hQkb funding bodies. We are currently exploring aspects of Micro-Satellite design.
2. All KEng-products will be assessed by a panel of IEEE-certified university mechanical engineering lecturers. This panel will declare a “best” and “second-based” and “most-doubtful” design. Their evaluations will be subjective: we only perform this part of the evaluation to ensure that hQkb members spend some non-trivial effort in exploring domain knowledge and improving their KEng-products.
3. Funding permitting, we will use trained independent software verification and validation professionals from NASA’s IV&V facility to assess the KEng-products.
4. External quality attributes will be used to assess the KEng-tool used and the KEng-product, as detailed below.

hQkb seeks to make definitive statements about what is knowable, rather than not make statements about what is unknowable. Hence, in order to produce objective quality assessments, certain quality attributes will be ignored. For example, under-defined quality attributes such as explainability, reliability, securability (integrity plus confidentiality), availability (of the system when required) will not be used. Further, quality attributes that require extensive human involvement (e.g. user usability) will be ignored. For more on quality attributes, see [Fenton & Pfleeger 1997, §9.1].

hQkb’s objective quality metrics will be restricted to that that can be determined using the **Design&Learn** framework (see §4.1) or some automatic tool (see §4.2).

4.1 Quality Assessment via Design&Learn

Given Design&Learn and a working hQkb system, quality measures can be determined for adaptability and resuability.

4.1.1 Adapatability

Adaption means changes to the system when requirements change. In the case of Design&Learn, such requirements would change our ideas, constraints, or assessment knowledge. That is, the more automatic support for the learn function within Design&Learn, the more adaptable the KEng-tool. Further, the more extensible the ideas, constraints, and assessment criteria within Design&Learn, the more adaptable it is.

4.1.2 Reusability

Three reuse measures follow from Design&Learn:

%-reused verbatim Suppose we access some supposedly reusable artifact before we start the Design&Learn loop.

The modifications made to that artifact before we exit Design&Learn is a measure of the reusableness of that artifact. Note that if we can reduce the artifact to some abstract search space A , then computing %-reused is easier. Within the portions built from reuse components, there should be a low ratio of new edges/old edges).

%-new construction: Decomposition and synthesis proponents might argue that a significant portion of the system would be based on reused components (i.e. a high ratio of edges from reused components/new edges).

Number of loops: Thirdly, a strong proponent of reuse might argue that if Design&Learn is initialized properly, the the first generated designs will be satisfying. That is, if we reuse the right knowledge, then we need only a few (1?) loops of Design&Learn.

4.2 Quality Assessment via Automatic Agents

This section describes some new research: automatic agents for quality attributes of discrete systems. It will be argued that such automatic agents may be able to infer many KBS quality metrics including:

- Predictability
- Maintainability
- Testability
- Controlability
- Observability
- Survivability

A pre-condition for many of these tests is that the testing agent can access G , a directed and-or graph representing the connections between literals in the knowledge base. This is not an unreasonable requirement. Many KEng tools already support some type of access to G (see the discussion of *search space reflection* at the end of [Menzies 1999]).

4.2.1 Maintainability and Testability

Maintainability: Maintenance means changes to the system when requirements are constant and the operating environment changes (or bugs appear). One measure of maintainability is, when considering changing “X”, can we find zones “Y” which are dependent on “X”? If we can, then after any change we can check what impact that change has on the rest of the system. This definition can be mapped into a search across G . Using G , we can compute points downstream of each test. Now, if we change some point within the system, we can limit our regression testing to only those tests which cross the effected regions. These points must be found within the intersection of the zone downstream of the change point and downstream of each test.

Testability: In conventional software engineering, an average definition of testability is the ability of the system’s interface to see errors. The more testable a KEng-tool, the more automatic support it offered for assessing testability. Several models of testability are computational; i.e. can be automatically assessed. For example:

Test suite generation: Zlatareva [1993] and Ginsberg [1987] discuss non-monotonic techniques for automatically generating test suites that cover all parts of a declarative KBS. G is sorted into consistent subsets. Each root of a subset is test case.

Mutation analysis: A *mutation* is a syntactically valid variant of a program. Mutations are random probes into the program space. Using large scale probing, it is possible to find sensitive spots where small changes have a major effect on the output of the program. Also, mutation analysis can be used to assess a test suite: it a test suite cannot detect the mutation, then it must be extended. Mutations can be done procedurally [Voas 1992] or declaratively [Menzies & Michael 1999] via structural alterations to G .

Minimizing test suite size: A repeated observation is that test suites often contain redundancies. That is, a test suite can be reduced without losing effectiveness. In one case study with program inputs, the reduction was down to 30% of the original size [Rothermel, Harrold, Ostrin & Hong 1998]. Several studies with program mutations report that a small number of mutants yield as much information as a large number (see the discussion in [Menzies & Michael 1999]).

Reproducibility: Menzies & Compton [1997] offer a model-based version of testability: the ability for a software model of “X” to reproduce known or desired behaviour of “X”. This definition may require a non-trivial extension to a KEng-tool. For example, this definition requires a library of known or desired behaviour. Special tools are required to store, generate, and execute such a library. For example, to generate this library from system invariants written in a first-order language, partially evaluate the invariants to generate instances. Each such instance is a test case. Also, if G is indeterminate, then a search across G implies making assumptions and keeping mutually exclusive assumptions in separate worlds of belief.

Distinguishability: Waugh, Menzies & Goss [1997] offers another specialization of testability: a system is **distinguishable** if the test results can distinguish a theory “X” from a revision to theory “X”. Distinguishable is slow to assess since it requires running thousands to millions of variations of “X”. However, it is automatable and, hence, is a candidate hQkb quality attribute.

4.2.2 Controlability, Observability, Survivability

Control theory discusses automatic agents for assessing continuous systems [Paulo & Arbib 1974]. The roots of an eigenvector of a system of equations can be queried to precisely determine the following quality attributes:

Controlability: A system is controllable if the system interfaces can be used to drive the system into a desired state.

Observability: A system is observable if the system interface can deduce what state the system is in.

Survivability: A system is survivable if, in any error state, the system can reach some safe state. That is, given meta-knowledge of what represents safe and unsafe operation, pathways exists from all unsafe states to safe states.

Analogous definitions can be offered for logical theories. Such an analysis would require:

- Access to G .
- Knowledge of which KBS literals come from system interfaces.
- Knowledge of what KBS states represent dangerous and safe situations.
- Fast theorem provers.

With these preconditions, then the following quality attributes can be defined for logical theories:

Logical observability: Any KBS state can be used to infer a unique set of interface literals.

Logical controlability: For all states, there exists a valid combinations of interface literals which can be used to infer that state,

Logical survivability: For all dangerous states, there exists a conjunction of a dangerous state and a valid combinations of interface literals which can be used to infer a safe state.

4.2.3 Predictability

Predictability is the ability to predict output states, given input states. A special type of predictability is **stability**: the ability to guarantee that all future states of the system will all be safe. Indeterminate AI algorithms are notoriously unpredictable. Predictability is an open research issue in KBS [Ishida 1989, Yip 1991], but of vital importance to organizations using autonomous, indeterminate, adaptive devices (such as NASA).

5 Conclusion

This document has:

- Briefly surveyed existing KEng methodologies and KEng evaluation projects. It was argued that prior KEng evaluation studies lacked quality measures that would be of interest to the KBS user community.
- Defined a new evaluation project with less technical biases than previous projects.
- Offered computational external quality attributes as a success measure.

hQkb is still in its formative stages. The purpose of this document is to form a community of KEng researchers interested in hQkb. This document is a success if it prompts extensive discussions amongst KEng researchers that significantly refine hQkb. Hence, further elaboration by this author is pointless. The real issue now is whether or not the international KEng community will endorse the hQkb project via any of:

- Discussions to refine the hQkb definition.
- Participation in the refined hQkb project.

Time will tell.

Acknowledgements

The comments of anonymous reviewers fixed a previous, overly complicated, version. Dialogues with Nigel Shadbolt (“where is the generality in your evaluation work?”) prompted this initiative. hQkb was crystalized by Josephson et al.’s [1998] SFV work and by the Sisyphus and HPKB communities who showed that international KEng researchers can work together to comparatively evaluate their work. Paul Compton and Bill Clancey convinced me KBS will always be changing and so must be continually evaluated. Greg Yost, Alun Preece, and Paul Cohen convinced me that such continual evaluations were technically possible. Discussions with SRL members helped clarify hQkb concepts. This work was partially supported by NASA through cooperative agreement #NCC 2-979.

References

- Alexander, C. [1964], *Notes on Synthesis of Form*, Harvard University Press.
- Alexander, C., Ishikawa, S., Silverstein, S., Jacobsen, I., Fiksdahl-King, I. & Angel, S. [1977], *A Pattern Language*, Oxford University Press.
- Angele, J., Fensel, D. & Studer, R. [1996], Domain and task modelling in mike, in A. S. et.al., ed., ‘Domain Knowledge for Interactive System Design’, Chapman & Hall.
- Barstow, D., Aiello, N., Duda, R., Erman, L., Forgy, C., Gorlin, D., Greiner, R., Lenat, D., London, P., McDermott, J., Nii, H. P., Politakis, P., Reboh, R., Rosenchein, S., Scott, A., van Melle, W. & Weiss, S. [1983], Languages and tools for knowledge engineering, in F. Hayes-Roth, D. Waterman & D. Lenat, eds, ‘Building Expert Systems’, Addison-Wesley, chapter 9, pp. 283–345.
- Benjamins, R. [1995], ‘Problem-solving methods for diagnosis and their role in knowledge acquisition’, *International Journal of Expert Systems: Research & Applications* 8(2), 93–120.
- Birnbaum, L. [1991], ‘Rigor Mortis: A Response to Nilsson’s ‘Logic and Artificial Intelligence’’, *Artificial Intelligence* 47, 57–77.
- Booch, G. [1996], *Object Solutions: Managing the Object-Oriented Project*, Addison-Wesley.
- Brachman, R., Gilbert, V. & Levesque, H. [1989], An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton, in J. Mylopoulos & M. Brodie, eds, ‘Readings in Artificial Intelligence and Databases’, Morgan Kaufmann, pp. 293–300.
- Brooks, R. [1991], ‘Intelligence without representation’, *Artificial Intelligence* 47, 139–159.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. [1996], *A System of Patterns: Pattern-Oriented Software Architecture*, John Wiley & Sons.
- Casady, G. [1996], Rationale in practice: templates for capturing and applying design expertise, in T. Moran & J. Carroll, eds, ‘Design Rationale: Concepts, Techniques, and Use’, Lawrence Erlbaum Associates, pp. 351–372.
- Chandrasekaran, B. [1983], ‘Towards a Taxonomy of Problem Solving Types’, *AI Magazine* pp. 9–17.
- Chandrasekaran, B., Johnson, T. & Smith, J. W. [1992], ‘Task structure analysis for knowledge modeling’, *Communications of the ACM* 35(9), 124–137.
- Clancey, W. [1985], ‘Heuristic Classification’, *Artificial Intelligence* 27, 289–350.
- Clancey, W. [1989], ‘The knowledge level reinterpreted: Modeling how systems interact’, *Machine Learning* 4(3/4), 285–293.
- Clancey, W. [1992], ‘Model Construction Operators’, *Artificial Intelligence* 53, 1–115.
- Cohen, P., Chaudhri, V., Pease, A. & Schrag, R. [1999], Does prior knowledge facilitate the development of knowledge-based systems?, in ‘AAAI’99’.
- Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D. & Burke, M. [1998], ‘The darpa high-performance knowledge bases project’, *AI Magazine* 19(4), 25–49.
- Compton, P. & Jansen, R. [1990], ‘A Philosophical Basis for Knowledge Acquisition.’, *Knowledge Acquisition* 2, 241–257.
- Crawford, J. & Baker, A. [1994], Experimental results on the application of satisfiability algorithms to scheduling problems, in ‘AAAI ’94’.
- Date, C. [1995], *An Introduction to Database Systems*, Vol. 6, Addison-Wesley.
- Easterbrook, S. [1991], Elicitation of Requirements from Multiple Perspectives, PhD thesis, Imperial College of Science Technology and Medicine, University of London. Available from <http://research.ivv.nasa.gov/~steve/papers/index.html>.

- Eriksson, H., Shahar, Y., Tu, S. W., Puerta, A. R. & Musen, M. A. [1995], 'Task modeling with reusable problem-solving methods', *Artificial Intelligence* **79**(2), 293–326.
- Etherington, D. & Reiter, R. [1983], On inheritance hierarchies with exceptions, in 'AAAI-83', pp. 104–108.
- Fenton, N. E. & Pfleeger, S. [1997], *Software Metrics: A Rigorous & Practical Approach*, International Thompson Press.
- Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J. & Nuseibeh, B. [1994], 'Inconsistency handling in multi-perspective specification', *IEEE Transactions on Software Engineering* **20**(8), 569–578.
- Fischer, G., Lemke, A., McCall, R. & Morch, A. [1996], Making argumentation serve design, in T. Moran & J. Carroll, eds, 'Design Rationale: Concepts, Techniques, and Use', Lawrence Erlbaum Associates, pp. 267–293.
- Forgy, C. [1982], 'RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem', *Artificial Intelligence* pp. 17–37.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. [1995], *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
- Gautier, P. O. & Gruber, T. R. [1993], Generating explanations of device behavior using compositional modeling and causal ordering, in 'AAAI-93', pp. 264–270.
- Gil, Y. & Melz, E. [1996], Explicit representations of problem-solving strategies to support knowledge acquisition, in 'Proceedings AAAI '96'.
- Ginsberg, A. [1987], A new Approach to Checking Knowledge Bases for Inconsistency and Redundancy, in 'Proc. 3rd Annual Expert Systems in Government Conference', pp. 102–111.
- Gruber, T. [1993], 'A translation approach to portable ontology specifications', *Knowledge Acquisition* **5**(2), 199–220.
- Inc., H. P. S. [1994], 'ithink 3.0.5'.
- Ishida, Y. [1989], Using global properties for qualitative reasoning: A qualitative system theory, in 'Proceedings of IJCAI '89', pp. 1174–1179.
- Jones, T. [1998], *Estimating Software Costs*, McGraw-Hill.
- Josephson, J., Chandrasekaran, B., Carroll, M., Iyer, N., Wasacz, B. & Rizzoni, G. [1998], Exploration of large design spaces: an architecture and preliminary results, in 'AAAI '98'. Available from <http://www.cis.ohio-state.edu/~jj/Explore.ps>.
- Kowalski, R. A. [1988], 'The early years of logic programming', *Communications of the ACM* **31**(1), 38–43.
- Laird, J. & Newell, A. [1983], A universal weak method: Summary of results, in 'IJCAI '83', pp. 771–773.
- Lenat, D. & Gutha, R. [1990], 'Cyc: A midterm report', *AI Magazine* pp. 32–59.
- Linster, M. [1992], A review of sisyphus 91 and 92: Models of problem-solving knowledge, in N. Aussenac, G. Boy, B. Gaines, M. Linser, J.-G. Ganascia & Y. Kordratoff, eds, 'Knowledge Acquisition for Knowledge-Based Systems', Springer-Verlag, pp. 159–182.
- Marcus, S. & McDermott, J. [1989], 'SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems', *Artificial Intelligence* **39**, 1–37.
- Marcus, S., Stout, J. & McDermott, J. [1987], 'VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking', *AI Magazine* pp. 41–58.
- Marques, D., Dallemagne, G., Kliner, G., McDermott, J. & Tung, D. [1992], 'Easy Programming: Empowering People to Build Their own Applications', *IEEE Expert* pp. 16–29.
- McIntosh, J. & McIntosh, R. [1980], *Mathematical Modeling and Computers in Endocrinology*, Springer-Verlag.
- Menzies, T. [1991], ISA Object PARTOF Knowledge Representation (part two)?, in B. Meyer, ed., 'Tools Pacific 4'. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/tools91.ps.gz>.
- Menzies, T. [1997], 'OO patterns: Lessons from expert systems', *Software Practice & Experience* **27**(12), 1457–1478. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/97probspatt.ps.gz>.
- Menzies, T. [1998a], 'Evaluation issues for problem solving methods'. Banff KA workshop, 1998. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/97eval>.
- Menzies, T. [1998b], 'Towards situated knowledge acquisition', *International Journal of Human-Computer Studies* **49**, 867–893. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/98ijhcs>.
- Menzies, T. [1999], 'Knowledge maintenance: The state of the art', *The Knowledge Engineering Review* **14**(1), 1–46. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/97kmall.ps.gz>.
- Menzies, T. & Clancey, B. [1998], 'Editorial, special issue on situated cognition', *International Journal of Human-Computer Studies* **49**.

- Menzies, T. & Compton, P. [1997], 'Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models', *Artificial Intelligence in Medicine* **10**, 145–175. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/96aim.ps.gz>.
- Menzies, T. & Mahidadia, A. [1997], Ripple-down rationality: A framework for maintaining psms, in 'Workshop on Problem-Solving Methods for Knowledge-based Systems, IJCAI '97, August 23.'. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/97rdra.ps.gz>.
- Menzies, T. & Michael, C. [1999], 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>'.
- Minsky, M. [1975], A framework for representing knowledge, in 'The Psychology of Computer Vision'.
- Moran, T. & Carroll, J. [1996], *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum Associates.
- Motta, E. & Zdrahal, Z. [1998], 'A library of problem-solving components based on the intergration of the search paradigm with task and method ontologies.', *International Journal of Human Computer Studies* **49**, 437–470.
- NASA [1991], 'CLIPS Reference Manual', Software Technology Branch, Lyndon B. Johnson Space Center.
- Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. & Swartout, W. R. [1991], 'Enabling technology for knowledge sharing', *AI Magazine* **12**(3), 16–36.
- Newell, A. [1982], 'The Knowledge Level', *Artificial Intelligence* **18**, 87–127.
- Newell, A. [1993], 'Reflections on the Knowledge Level', *Artificial Intelligence* **59**, 31–38.
- Nilsson, N. [1991], 'Logic and artificial intelligence', *Artificial Intelligence* **47**, 31–56.
- Noy, N. F. & Hafner, C. [1997], 'The state of the art in ontology design: A survey and comparative review', *AI Magazines* pp. 53–74.
- Nuseibeh, B. [1997], 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)', IEEE CS Press., pp. 164–169.
- Paulo, L. & Arbib, M. [1974], *System Theory*, W.B. Saunders.
- Pearl, J. & Korf, R. [1987], 'Search techniques', *Ann. Rev. Comput. Sci. 1987* **2**, 451–67.
- Preston, P., Edwards, G. & Compton, P. [1993], A 1600 Rule Expert System Without Knowledge Engineers., in J. Leibowitz, ed., 'Second World Congress on Expert Systems'.
- Richards, D. & Menzies, T. [1998], Extending the sisyphus iii experiment from a knowledge engineering task to a requirements engineering task, in 'Banff Workshop on Knowledge Acquisition'. Available from <http://www.cse.unsw.edu.au/~timm/pub/docs/98kawre.ps.gz>.
- Rittel, H. [1972], Second generation design methods, in 'Design Methods Group 5th Anniversary Report: DMG Occasional Paper, 1, 5-10.'. Also in N. Cross (Ed.) *Developments in design methodology*, (pp. 317-327). Chichester: Wiley & Sons, 1984.
- Rosenbloom, P., Laird, J. & Newell, A. [1993], *The SOAR Papers*, The MIT Press.
- Rothermel, G., Harrold, M., Ostrin, J. & Hong, C. [1998], An empirical study of the effects of minimization on the fault-detection capabilities of test suites, in 'Proceedings of International Conference on Software Maintenance '98', pp. 34–43. Available, via email request, from harrold@cis.ohio-state.edu.
- Schon, D. [1983], *The Reflective Practitioner*, Harper Collins/ Basic Books.
- Schreiber, A. T. & Birmingham, W. P. [1996], 'The sisyphus-vt initiative', *International Journal of Human-Computer Studies* **44**(3/4).
- Schreiber, A. T., Wielinga, B., Akkermans, J. M., Velde, W. V. D. & de Hoog, R. [1994], 'Commonkads. a comprehensive methodology for kbs development', *IEEE Expert* **9**(6), 28–37.
- Selman, B., Levesque, H. & Mitchell, D. [1992], A new method for solving hard satisfiability problems, in 'AAAI '92', pp. 440–446.
- Shadbolt, N., O'Hara, K. & Crow, L. [2000], 'The experimental evaluation of knowledge acquisition techniques and methods: History, problems and new directions', *International Journal of Human-Computer Studies* . (to appear).
- Shalin, V., Geddes, N., Bertram, D., Szczepkowski, M. & Dubois, D. [1997], Expertise in dynamic, physical task domains, in P. Feltovich, K. Ford & R. Hoffman, eds, 'Expertise in Context', MIT Press, chapter 9, pp. 195–217.
- Simon, H. [1969], *The Science of the Artificial*, MIT Press.
- Steels, L. [1990], 'Components of Expertise', *AI Magazine* **11**, 29–49.
- Suchman, L. [1993], 'Response to vera and simon's situated action: A symbolic interpretation', *Cognitive Science* **17**, 71–75.

- Swartout, B. & Gill, Y. [1996], Flexible knowledge acquisition through explicit representation of knowledge roles, in '1996 AAAI Spring Symposium on Acquisition, Learning, and Demonstration: Automating Tasks for Users'.
- Tecuci, G. [1998], *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*, Academic Press.
- Ullman, J. [1988], *Principles of Database and Knowledge-base Systems*, Computer Science Press.
- Uschold, M. & Gruninger, M. [1996], 'Ontologies: Principles, methods, and applications', *The Knowledge Engineering Review* **11**(2), 93–136.
- van Heust, G., Schreiber, A. T. & Wielinga, B. [1997], 'Using explicit ontologies in kbs development', *International Journal of Human Computer Studies* **45**, 183–292.
- Voas, J. [1992], 'Pie: A dynamic failure-based technique', *IEEE Transactions of Software Engineering* **18**(2), 717–727.
- Wagh, S., Menzies, T. & Goss, S. [1997], Evaluating a qualitative reasoner, in A. Sattar, ed., 'Advanced Topics in Artificial Intelligence: 10th Australian Joint Conference on AI', Springer-Verlag.
- Webber, H. R. . M. [1973], 'Dilemmas in a general theory of planning', *Policy Sciences* **4**, 155–169.
- Wielinga, B., Schreiber, A. & Breuker, J. [1992], 'KADS: a Modeling Approach to Knowledge Engineering.', *Knowledge Acquisition* **4**, 1–162.
- Williams, B. & DeKleer, J. [1991], 'Qualitative reasoning about physical systems: a return to roots', *Artificial Intelligence* **51**, 1–9.
- Winograd, T. [1975], Frame representations and the declarative/procedural controversy, in 'Readings in Knowledge Representation', Morgan Kaufman, pp. 185–210. Also available R.J. Brachmann and H.J. Levesque (eds), *Readings in Knowledge Representation*, Morgan Kaufmann, Palo Alto, 1985.
- Yip, K. [1991], 'Understanding complex dynamics by visual and symbolic reasoning', *Artificial Intelligence* **51**, 179–221.
- Yost, G. [1994], Implementing the Sisyphus-93 task using SOAR/TAQL, in B. Gaines & M. Musen, eds, 'Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop', pp. 46.1–46.22.
- Yost, G. & Newell, A. [1989], A Problem Space Approach to Expert System Specification, in 'IJCAI '89', pp. 621–627.
- Zlatareva, N. [1993], Distributed verification and automated generation of test cases, in 'IJCAI '93 workshop on Validation, Verification and Test of KBs Chambery, France', pp. 67–77.

A Primary Modeling Constructs

This section lists the primary modeling constructs seen in the software engineering and knowledge engineering literature.

Axioms: Assertions about a particular domain;

Single general-purpose inference engines ⁵

PSMs: Libraries of declarative representations of inferencing clichés⁶; e.g. the diagnosis PSM in Figure 8.

Ontologies: Libraries of abstracted data types seen in different domains⁷. Ontologies model the background data within a domain and have been called “an explicit specification of a conceptualisation” [Gruber 1993]. PSMs and ontologies are linked: ontologies define the data types required by a PSM to execute in a domain. For example, the diagnosis PSM of Figure 8 needs (e.g.) *complaints* and *observables* to execute.

The following three constructs are defined for completeness sake. While exceptions exist, they are not the usual modeling construct in KEng:

Maths: A non-trivial range of systems can be modeled as quantitative or qualitative equations [McIntosh & McIntosh 1980, Williams & DeKleer 1991].

Data Modeling: A modeling approach whose target is a set of relational database tables [Date 1995].

Procedures: Libraries storing know-how represented as procedural code.

Objects: Combinations of data with the functions that process that data into one encapsulated unit. Units may exist in a hierarchy and sub-units inherit functions and data definitions from super-units. Proponents of objects modeling use objects as a tool to specify higher-order structures such as blackboards [Buschmann et al. 1996]. However, as a knowledge representation tool, standard object-oriented languages have major restrictions [Menzies 1991].

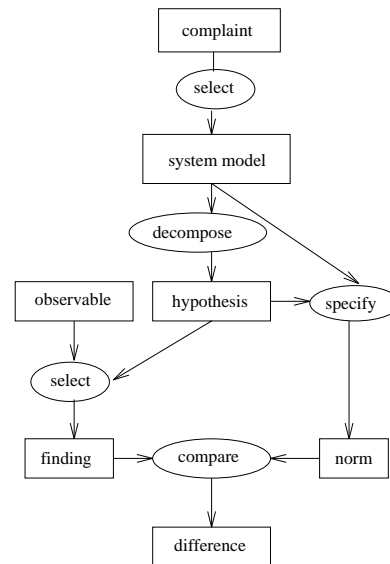


Figure 8: A PSM for diagnosis; ovals=functions, rectangles=data types.

⁵E.G. Prolog [Kowalski 1988], OPS5 [Forgy 1982], SOAR [Laird & Newell 1983, Rosenbloom et al. 1993], PSCM [Yost & Newell 1989], GSAT [Selman, Levesque & Mitchell 1992], ISAMP [Crawford & Baker 1994],...

⁶SPARK/BURN/FIREFIGHTER (SBF) [Marques, Dallemagne, Kliner, McDermott & Tung 1992]; generic tasks [Chandrasekaran et al. 1992]; configurable role-limiting methods [Swartout & Gill 1996, Gil & Melz 1996]; model construction operators [Clancey 1992]; CommonKADS [Wielinga et al. 1992, Schreiber et al. 1994]; the PROTEGE approach [Eriksson et al. 1995]; components of expertise [Steels 1990]; MIKE [Angele, Fensel & Studer 1996]; TINA [Benjamins 1995].

⁷E.G. [Lenat & Gutha 1990, Gruber 1993, Neches, Fikes, Finin, Gruber, Patil, Senator & Swartout, 1991, Uschold & Gruninger 1996, Noy & Hafner 1997]. In recent years, ontologies have also appeared in research into software architectures and design patterns [Menzies 1997].