AI Magazine Volume 6 Number 3 (1985) (© AAAI)

LETTERS

On "Learning Language"

Editor:

I was dismayed by the inclusion of William Katke's article ("Learning Language Using A Pattern Recognition Approach," Spring 1985). Usually you do an excellent job of representing "the current state of the art in Artificial Intelligence" (to quote your Editorial Policy), but I consider this article an exception.

First of all, although the article claims to be on "Learning Language," what it presents is at best a knowledge-free approach to learning syntax. I saw no evidence that the induced syntax is useful for anything, and good reasons to believe that it is not, such as the unmnemonic category names and the intrinsic limitations of finite state grammars.

Second, this kind of stuff has been done before, and it didn't work too well then either; for a useful overview of the field and pointers into the literature, see the article on "Grammatical Inference" in Volume 3 of *The Handbook of Artificial Intelligence,* (ed. Cohen and Feigenbaum).

Third, even several years ago AI researchers were building systems that had more to do with real language learning. For example, Anderson's LAS (*Cognitive Science* 1:2, 1977) induced the mapping between strings in a context-free language and semantic network representations of their meaning. Granger's FOUL-UP (IJCAI-77) and Carbonell's POLITICS (*Cognitive Science* 2:1, 1978) addressed the problem of learning the meaning of new words from context.

Fourth, the whole pattern recognition approach ignores the central issues in language learning, such as representation, semantics, and domain knowledge. Even the section on Future Research talks about "setting parameters" instead of addressing these issues. The whole point of AI is to escape from the fruitlessness of syntactic pattern recognition by bringing knowledge to bear.

Fifth, besides having little to do with the rest of the paper, the Conclusion manages to place Schank in the "logic" camp. This is pretty funny considering what he had to say about logic in the Great Debate with McCarthy at AAAI83.

Katke's project appears to be a classic example of "climbing a tree to get to the moon." It is distressing to see so much misdirected effort.

Jack Mostow
Department of Computer Science
Rutgers University
Hill Center—Busch Campus
New Brunswick, New Jersey 08903

Editor:

Jack Mostow is reading things into the article that are not in it. The article is not recommending a pattern recognition approach in place of a knowledge based approach. We are exploring learning by pattern recognition of knowledge for knowledge based systems. This is clearly stated in the first paragraph of the article.

Several of Mostow's remarks indicate lots of things not done in the article. The purpose of the article was not to explore the entire domain or solve all the problems. The intent is this: "Here is a simple but powerful learning algorithm. Future research needs to be done to demonstrate its usefulness. Intuition suggests this algorithm and others like it will be useful." There would be little progress in science if people did not share their intuitions.

Mostow points out that this kind of stuff was done once before and it didn't work. It's only fair to also point out that no approach has yet solved the language processing problem. Inferencing did not find practical applications until computers were big enough and fast enough to support that kind of processing. Learning by pattern recognition was tried even before inferencing. It had little chance of success without the necessary info structure. Expert systems and knowledge based approaches to processing natural languages are part of that info structure.

A better way to characterize knowledge acquisition research would be to say if knowledge based approaches are the rockets that will get us to the moon then learning techniques are the pumps that will fill the fuel tanks. Pattern recognition will be one of those pumps.

William Katke
IBM
Page Mill Road
Palo Alto, CA 94304

Editor:

Design, like system development in AI, is an incremental and an exploratory process.

I was surprised by the recent article "Towards better models of the design process" (Mostow, 1985) that introduced "the key research problem in AI-based design" as the development of better models of the design process. The article, a report based on the Rutgers Workshop on Knowledge Based Design Aids, presented "some of the most important ideas emerging from current AI research on design, especially ideas for better models of design".

While the report did a good job of laying out the possibilities in terms of numbered ideas (Idea #i) and issues (Issue #i), there was no mention of design in AI being an incremental, fundamentally exploratory process. The ideas and issues presented were firmly focused on a conventional view of the design process—a view I can caricaturize as the SPIV methodology:

$$\text{Specify} \longrightarrow \text{Prove} \longrightarrow \text{Implement} \longrightarrow \text{Verify}$$

Variants and refinements of the SPIV methodology were all that was reported. The original workshop may have been directed only at the use of AI to assist in conventional design problems. It may have purposely ignored the question of a design methodology for AI itself.

In either event it does not matter, I would still maintain that a rather different and important paradigm for design was omitted. The paradigm, which has no name although phrases like Run-Understand-Debug-Edit (RUDE) begin to capture the essential cycle involved, is, I claim, a viable but thoroughly neglected alternative for both AI system design and more traditional design processes (architectural design, for example, see Bazjanac, 1974). There may also be no real distinction to be made there either; again, it does not matter for my present argument. In my opinion, an important class of design processes appeared to be ignored.

Let me try to sketch out this alternative paradigm and indicate my reasons for believing in its importance with respect to design in AI (and by implication, its importance to design in general).

Rather than verifying an implementation of a reasonably complete specification, the procedure in AI can be better described as incrementally developing an adequate approximation to some incompletely specified function. One can agree with this coarse distinction but claim that its significance is not the exposure of an alternative paradigm for design; instead its significance is a reflection of the "hacking syndrome" that is endemic in AI. When the field reaches maturity, a state whose advent will be hastened by the current surge of interest in commercial AI, the RUDE paradigm will be replaced by a SPIV-based methodology. AI system designers will then have become software engineers distinguished only by the complexity of their domain.

This view, I argue, is at best premature, and at worst totally wrong: it may be that a RUDE-based methodology is an approach well-suited to the nature of AI problems. That is not to say that typical manifestations of RUDE techniques are satisfactory—far from it. It is only to say that the general nature of the RUDE paradigm may be more appropriate as a basis for AI design than is the SPIV paradigm. The RUDE paradigm is in dire need of development and many of the ideas and issues described by Mostow will contribute to such development.

Complex and ill-structured problems are the domain of AI, and the design of adequate implementations of such problems necessarily appears to be an incremental, evolutionary, exploratory process. Instead of striving for complete specifications and the verification of proposed implementations, we should concentrate more on incremental development of specifications as a result of assessment of performance. This in turn depends on development of abstraction techniques (in order to obtain intellectually manageable specifications from behaviorally interesting programs), and techniques of design-for-change.

Sandewall (1978), for example, has raised this last point with respect to structured programming and AI. The loose conglomerate of rules and guidelines known as structured programming is an all-at-once technique, which is typical of work within the SPIV paradigm. Given a complete specification of some problem, how can we implement it such that the resulting object is maximally transparent to humans? That is the question that structured programming seeks to answer. If any changes are introduced they are introduced into the specification, which is then reimplemented (at least that is the theory, the practice is all too often some manifestation of RUDE techniques of the worst kind).

By way of contrast, the RUDE paradigm suggests a need for some analogous but different concept—Sandewall calls it "structured growth". Such a concept would involve rules and guidelines for incrementally adding structured code in a way that maintains overall clarity of structure, and for altering parts of a structured object, again in a way that preserves its perspicuity.

I can single out two characteristics of AI problems that argue for a RUDE, rather than a SPIV, paradigm for the design process.

First, many AI problems are highly context sensitive, and the relevant context is not easily circumscribed. Worse than that, many AI problems exhibit tightly-coupled context sensitivity: intelligent answers to such problems are highly dependent on contextual information, *i.e.,* information external to the particular problem at hand. Thus, for example, the meaning of a sentence may be minimally dependent on the actual words that constitute the sentence. The interpretation of an image may also have little dependence on the actual image data itself. The best explanation of the reasoning behind some decision can be more dependent on who wants the explanation, and why, than on the particular decision itself, and so on.

A succinct and complete context-free specification of such problems thus appears to be an unrealistic goal. (Although there is always hope that such apparently tightly-coupled, context sensitive problems can be decomposed into a collection of more manageable, fairly modular subproblems.)

An example of a non-tightly-coupled AI problem is chess: the next best move for any board configuration is (almost) totally dependent upon the particular configuration and independent of any contextual considerations.

The second characteristic that I will mention in support of my argument is adaptivity, which translates into a need for machine learning in AI. A number of observers (see for example Schank, 1983, and Samuel, 1983) see machine learning, despite years of neglect, as critical to the design of AI systems. I have argued at length and in detail in Partridge (1985) for this and for most other points that I have had to deal with very cursorily here. Reasons why I believe that a non-trivial machine learning capability must be accommodated in a paradigm for the design of AI systems, and why such a belief implies the RUDE rather than the SPIV paradigm, are:

(i) Everyone is different, if an AI system is to behave even reasonably intelligently, it cannot neglect this fact.

(ii) Any one person is different at different times: a major role of AI systems is to impart knowledge to people, and if it does not respond to the changes it induces, it will have failed—it will not be AI.

(iii) The empirical world is a rapidly and subtly changing place (quite apart from the people in it); an AI system to remain an AI system must keep abreast of the relevant changes.

The discovery of robust and generalized learning algorithms would perhaps enable the necessary adaptivity to be accommodated within the SPIV paradigm. But until such time as we find these learning algorithms (and I don't think that many would argue that such algorithms will be available in the foreseeable future) we must face the prospect of systems that will need to be modified, in non-trivial ways, throughout their useful lives. Thus incremental development will be a constant feature of such software and if it is not fully automatic then it will be part of the human maintenance of the system. I am, of course, not suggesting that the products of say architectural design (*i.e.*, buildings) will need a learning capability. Nevertheless, a final fixed design, that remains "optimal" in a dynamically changing world, is a rare event.The similarity between AI system development and the design of more concrete objects is still present, but it is, in some respects, rather tenuous I admit.

In sum, the first characteristic, tightly-coupled context sensitivity, implies that design involves incremental development of an adequate specification. And the second characteristic, adaptivity, implies incremental development throughout the life of the system.

I have discussed various approaches to, and aspects of, a disciplined RUDE-based methodology for AI (*e.g.*, Partridge, 1978, 1981, and 1984). I am clearly a believer, but is anybody else? I find the neglect of incremental design techniques all the more surprising in view of the current concerns with expert systems. The designers of such systems must deal with incomplete knowledge bases (no one seriously suggests that a complete knowledge base is possible), and with incrementally updating these knowledge bases. They need logics for inferencing from incomplete knowledge (as proposed for example by Levesque, 1984), and they need a paradigm for incrementally updating the knowledge without generating an unmanageable tangle—*i.e.*, expert system designers seem to have a clear need for a RUDE-based paradigm.

Hewitt (1985) argues for what I take to be some RUDE-like paradigm for "developing the intelligent systems of the future". He discusses the problems of continuous change and evolution, and the need to accommodate necessarily incomplete information, which implies, he says, exploration rather than searching—the traditional approach in AI.

One can in fact take this argument right into the camp of traditional software engineering. There is a growing realization in software engineering (see for example Giddings, 1984) that a fundamental feature of such software is that it grows and changes—or that it should. This being the case, the static SPIV paradigm is being seriously questioned as to its appropriateness as the standard to be achieved.

De Millo, Lipton, and Perlis (1979) have argued fairly convincingly that the possibility of ever formally verifying real-world programs (as opposed to well-defined abstract functions) is vanishingly small. And even if verification were possible it would not contribute very much to the development of production software. Hence "verifiability must not be allowed to overshadow reliability. Scientists should not confuse mathematical models with reality."

AI is perhaps not so special, it is rather an extreme and thus certain of its characteristics are more obvious than in conventional software applications. Thus the SPIV methodology may be inappropriate for an even larger class of problems than those of AI.

I have raised all these points not to try to deny the worth of Mostow's ideas and issues concerning the design process, but to make the case that such endeavors should also be pursued within a fundamentally incremental and evolutionary framework for design. The potential of the RUDE paradigm is deserving of more attention than it is customarily accorded. It is not yet obvious that the design of AI systems should aspire to a fundamentally SPIV-type methodology—it is still an open question.

References

Bazjanac, V. (1974) Architectural design theory: Models of the design process. In W. R. Spillers (Ed.), *Basic Questions of Design Theory.* Amsterdam: North-Holland.

De Millo, R. A., Lipton, A. J., & Perlis, A J. (1979) Social processes and proofs of theorems and programs. *Communications of the ACM* 22(5):271-280.

Giddings, R. V (1984) Accommodating uncertainty in software design. *Communications of the ACM* 27(5):428-43.

Hewitt, C (1985) The challenge of open systems. *Byte.* April, pp. 223-242.

Levesque, H J (1984) The logic of incomplete knowledge bases, in on conceptual modelling M L. Brodie, J Mylopoulos, & J. W. Schmidt (Eds), NY:Springer-Verlag

Partridge, D. (1978) *A philosophy of "wicked" problem implementation* Proceedings of the AISB/GI Conference, Hamburg, 238-247

Partridge, D. (1981) "Computational theorizing" as the tool for resolving wicked problems. IEEE Trans Systems, Man, and Cybernetics, SMC-*11*, No. 4, *318-322.*

Partridge, D. (1984) *What's in an AI program?* Proceedings of ECAI84, 669-673. Pisa, Italy,

Partridge, D. (1985) *Artificial intelligence and the future of software engineering* Chichester, UK: Ellis Horwood (forthcoming)

Mostow, J. (1985) Toward better models of the design process. *AI Magazine* 6(1): 44-57.

Samuel, A. L. (1983) *AI, where has it been and where is it going?* IJCAI-83, 1152-1157

Sandewall, E. (1978) Programming in an Interactive Environment: the "LISP" experience. *Computing Surveys* 10(1):35-71.

Schank, R. C. (1983) The current state of AI: One man's opinion. *AI Magazine*, Winter-Spring, 1983, pp. 3-8.

Derek Partridge
Computing Research Laboratory
New Mexico State University
Las Cruces NM 88003

Response to Derek Partridge

Editor:

On first reading Partridge's comments, I found myself agreeing with many of them; in fact, I thought I had made some of the same points in the article, though perhaps with less emphasis. In particular, I agree that exploration and learning are important, and said as much in the section on "Investigating the role of learning in design." However, his emphasis on the iterative nature of design is well-taken.

Before proceeding further, I'd better clean up a possible confusion between two closely related concepts. A design "model" describes how design is done, while a design "methodology" prescribes how it ought to be done. My article was concerned with improving models of design by explicitly representing various aspects of the process that at present are imperfectly understood, not with advocating a particular design methdology. However, aside from his comment that "an important class of design processes appeared to be ignored," which can be taken as a criticism of a design model, Partridge largely addresses issues of methodology. Of course, better models may suggest improved methodologies, and the distinction is further blurred by the current movement toward knowledge-based design systems whose added leverage comes from making more aspects of the design process explicit.

One of these aspects is the intended behavior of the designed artifact. Obviously any comprehensive model of the design process should include a description of this behavior, *i.e.*, a specification. Where Partridge and I apparently disagree is on the proper role of such a specification in a design methodology. Partridge's RUDE methodology focusses on design iteration, but neglects the role of a specification other than the program itself.

At the other extreme he places SPIV, a straw man that I never advocated in the first place. Partridge's feedback-free caricature of the SPIV methodology apparently constructs a complete specification and then proves, implements, and verifies it without modification, thus neglecting the iterative nature of the specification process and the specification changes resulting from implementation and use. Contrary to Partridge's diagram, a specification cannot be "proved," at least not in the sense of verifying a program against a formal description of what it is supposed to do; for a specification, there is no such higher level description. At best, a specification can be subjectively validated against the designer's intent; methods for assisting validation include natural language paraphrasing, static analysis, symbolic execution, and experimenting with a prototype, but are necessarily incomplete, since the designer's complete intent is inaccessible to the machine (and often to the designer). Moreover, as Partridge points out, "valid" designs are invalidated by inevitable changes in the designer's intent or the system's environment. Also, the specification may need to be changed as a result of the methods used to implement it (Swartout & Balzer, 1982).

Unfortunately, my article seems to have taken these points too much for granted in its discussion of specification changes and reimplementation. The companion report (Mostow, 1984) was more explicit: "(Scherlis observed that) although a derivation is an idealized design history of the implemented code, the actual design process need not be a linear progression of commitments leading from specification to implementation... a derivation may be designed by patching it, not just by successively extending it."

The article gives several compelling reasons for including such a specification or idealized design history as an explicit part of a design methodology. The one most relevant here has to do with assisting design exploration: if reimplementation is the inner loop of design exploration, automating it should free the designer to explore alternative designs more easily. "If the specification is modified, it may be possible to reimplement it by replaying the derivation of the original implementation. A specification change may necessitate patching the transformation sequence in places where the original design decisions are no longer appropriate. However, this is much cleaner than patching the end product, where the effects of the revisions may be widespread" (page 46, citations omitted). I will call this specification-based methodology COURTEOUS, for "Changes Obtained by Using Replay of Transformations to Enforce Oft-Updated Specifications."

The differences between COURTEOUS and RUDE can be clarified by drawing an analogy between implementation and compiling. Modifying an implementation directly (*i.e.*, RUDEly) is like patching compiled code. In contrast, the COURTEOUS approach of modifying a specification and replaying the derivation corresponds to editing a high-level program and recompiling it, and should be easier for the same reasons. For example, if the effects of a single specification change or design decision are distributed throughout the implementation, it is easier to change one part of the derivation than to make patches all over the implementation. This distribution effect explains why patched code is hard to understand and maintain, and why "rules and guidelines for incrementally adding structured code in a way that maintains overall clarity of structure" are not always feasible.

To continue the analogy between implementing and compiling, the process advocated by Partridge for abstracting "intellectually manageable specifications from behaviorally interesting programs" corresponds to decompiling, *i.e.*, the notoriously difficult problem of inferring function from structure. A solution would be useful, but it seems much easier to support the implementation process than to invert it.

After describing the RUDE paradigm, Partridge explains why he thinks it is especially well-suited to the design of AI systems. I did not follow all of his arguments here.

In particular, I do not see how the context-sensitivity of explanation argues against the use of specifications. The various contextual factors are simply additional inputs to the explanation process. Admittedly, this information is difficult to represent, but this seems irrelevant. Certainly realizing that a computation requires an additional input is a typical specification change. I think the basic argument is that the tight coupling between an AI system and the environment in which it operates implies that the implementation of the system must proceed in tandem with developing a model of the environment, which is part of the specification. This may argue against SPIV, but not against COURTEOUS.

Partridge sees "machine learning, despite years of neglect, as critical to the design of AI systems." As a researcher in machine learning, I agree on its importance, and would like to emphasize that the years of neglect have been over for some time; witness the 1980, 1983, and 1985 International Workshops on Machine Learning, and the flurry of recent papers.

Partridge cites three reasons why AI systems should learn.

- Point (i) involves individual differences, but he fails to relate them to machine learning.
- Point (ii) says that an interactive AI system should update its user model (I agree) and is not AI if it doesn't (this is overstating the case).
- Point (iii) says that an AI system should update itself in the face of a changing environment. (Lenat *et al.*, 1979) makes the same point and offers some solutions.

Unfortunately, Partridge fails to spell out why any of these points favors the RUDE approach.

I agree that adaptive systems will require incremental development; in fact this is true of virtually all software, not just AI systems. However, I disagree that expert system designers need a RUDE-based "paradigm for incrementally updating the knowledge without generating an unmanageable tangle." On the contrary, this tangle is often a symptom of the patching-compiled-code syndrome. In this case, the "code" consists of rules compiled by integrating diverse sorts of expertise. The COURTEOUS solution is to factor these different sources of knowledge apart, and explicitly record them and the process by which they are combined. Structuring the design process in this way should enable the resulting expert system to give better explanations of its behavior and enhance its maintainability (Swartout, 1983; Neches *et al.*, 1984).

The COURTEOUS approach does not come for free: Making specifications and derivations explicit imposes a high overhead on the initial design process. The payoff can be expected to come during subsequent redesign. The overhead of increased formalization may not be worth it for a throw-away experimental AI system used only by its creator, but should eventually be amortized for a complex, long-lived system by reducing the cost of maintaining it.

As Partridge observes, current practice tends to patch existing implementations rather than reimplement. This can be attributed to the cost of manual reimplementation. As automated replay tools are developed that lower the cost of reimplementation closer to recompilation, the COURTEOUS approach should become more feasible.

References

Lenat, D , Hayes Roth, F , & Klahr P. (1979) Cognitive economy in artificial intelligence systems IJCAI-6, 531-536.

Mostow, J. (1984) Rutgers workshop on knowledge-based design. SIGART Newsletter (90), October 1984, 19-32.

Neches, R , Swartout, W., & J. Moore (1984) Enhanced maintenance and explanation of expert systems through explicit models of their development. Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, Denver, Colorado, December 1984

Swartout, W. (1983) XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence* 21(3):285-325 Also available as ISI/RS-83-4.

Swartout, W., & R. Balzer (1982) On the inevitable intertwining of specification and implementation. *CACM*, July 1982, 438-440

Jack Mostow
Department of Computer Science
Rutgers University
Hill Center—Busch Campus
New Brunswick, NJ 08903

Editor:

We believe that a comment in the letter "AAAI-84 Profile" which appeared in the Spring, 1985 issue of the AI Magazine is inappropriate in a professional journal. The statement "some half of [the women present at AAAI-84] wore no wedding ring" implied that a segment of female

AAAI members attending a professional meeting was potentially "available." By publishing the letter without comment you tacitly encourage viewing female members, not as fellow members and colleagues, but as objects of a very unprofessional sort of attention.

Phyllis Koton Deborah Estrin
Sharon Gray Rivka Ladin
Mike Eisenberg
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge Massachusetts 02139

Gavin Duffy Bonnie Dorr
John Batali Dalvid A. Levitt
Mark Shirley Robert Giansiracusa
Fanya S. Montalvo
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Square
Cambridge, Massachusetts 02139

Kent M. Pitman V. Ellen Golden
Symbolics Inc.
Four Cambridge Center
Cambridge, Massachusetts 02142

Editor:

The AI Magazine should have a humor section to liven it up, just a smidgen.

The attached just came off the e-mail after several different people got into the act after Craig Reynolds's original answer. I think it is funny, and it concerns lisp-machines and Lisp and so forth.

Bob Stone
Symbolics, Inc.
11 Cambridge Center
Cambridge, MA 02142

What is the bridge mailing list?

OK, it was the mailing list used during the work that the precursor of the Graphics Division did for Paramount Pictures for the feature film STAR TREK III: The Search for Spock. We created 24 fps video elements which were recorded and played back through monitors on the set of the bridge of the Enterprise and several other ships. These 24 hz monitors (kept in sync with the film camera) were then just photographed along with the rest of the set to provide the appearence of a working space ship bridge.

"Click on [Warp 5], Mr. Sulu."

"Sir! The controls aren't responding!"

"Type ⟨Function⟩ control-⟨Clear Input⟩!"

"Captain, the Screen Manager canna take much more o' this!"

"[Emergency Break]!"

"It's no good sir, we're in the Cold Load Stream. . ."

"Message from Star Fleet Headquarters, Captain, on Serial Line Zero."

"Put it on Lisp Listener 5, Uhura."

"Greetings, Captain Kirk. SCRC-Stony-Brook is up for service now. Just thought you might want to know."

"Uhura, deexpose Lisp Listener 5. Yes, Spock, what is it now?"

"We only have one megaword left before we lose our last chance to enable incremental garbage collection, Sir."

"What about the ephemeral GC, Spock?"

"It can't keep up at this speed, Sir. As long as we remain above Warp Three, the New Dilithium System conses like a bear."

"Your recommendations?"

"Cut back to impulse power and do a GC-immediate, sir."

"I'm afraid that's not possible. We have to get to Meta Beta Three with this shipment of cartridge tapes before their next full FS dump, or millions of innocent files may die."
