Marco Valtorta. Settembre 1989
(earlier copy lost)

Does AI have a methodology different from Software Engineering?

MCCS-86-53

Derek Partridge and Yorick Wilks

Computing Research Laboratory
New Mexico State University
Box 3CRL
Las Cruces, NM 88003

ABSTRACT

The paper argues that the conventional methodology of software engineering is inappropriate to AI, but that the failure of many in AI to see this is producing a Kuhnian paradigm "crisis". The key point is that classic software engineering methodology (which we call SPIV: Specify-Prove-Implement-Verify) requires that the problem be circumscribable or surveyable in a way that it is not for areas of AI like natural language processing. In addition, it also requires that a program be open to formal proof of correctness. We contrast this methodology with a weaker form SAT (complete Specification And Testability - where the last term is used in a strong sense: every execution of the program gives decidably correct/incorrect results) which captures both the essence of SPIV and the key assumptions in practical software engineering. We argue that failure to recognize the inapplicability of the SAT methodology to areas of AI has prevented development of a disciplined methodology (unique to AI, which we call RUDE: Run-Understand-Debug-Edit) that will accommodate the peculiarities of AI and also yield robust, reliable, comprehensible, and hence maintainable AI software.

*The Computing Research Laboratory is principally funded by the New Mexico State Legislature,
administered by its Scientific and Technical Advisory Committee
as part of the Rio Grande Research Corridor.*

Introduction: Kuhnian paradigms in AI

Is it helpful or revealing to see the state of AI in, perhaps over-fashionable, Kuhnian (Kuhn, 1962) terms? In the Kuhnian view of things, scientific progress comes from social crisis: there are pre-paradigm sciences struggling to develop to the state of "normal science" in which routine experiments are done within an overarching theory that satisfies its adherents, and without daily worry about the adequacy of the theory.

At the same time, there will be other scientific theories under threat, whose theory is under pressure from either disconfirming instances or fundamental doubts about its foundations. In these situations, normal science can continue if the minds of adherents to the theory are closed to possible falsification until some irresistible falsifying circumstances arise, by accretion or by the discovery of a phenomenon that can no longer be ignored.

There is much that is circular in this (the notion of "irresistible" for example) and there may be doubts as to whether AI is fundamentally science or engineering (we return to this below). But we may assume, for simplicity, that even if AI were engineering, similar social descriptions of its progress might apply (see Duffy 1984).

Does AI show any of the signs of normality or crisis that would put it under one of those Kuhnian descriptions, and what would follow if that were so? It is easy to find normality: the production of certain kinds of elementary expert system (ES) within commercial software houses and other companies. These work well enough for straightforward applications, yet doubts about their extensibility are widespread.

Crisis and pathology are even easier to find, and our diagnosis in brief is this: normal AI is impeded by the fact that, whether they are aware of it or not, a wide range of AI's academic practitioners are struggling to conform to another paradigm because they suspect their own is inadequate. On our view the natural normal paradigm of AI is RUDE (Run-Understand-Debug-Edit). But pressure and crisis come from the SPIV (Specify-Prove-Implement-Verify) methodology, and its weaker version SAT (complete Specification-And-Testability of program behavior). The nature of this crisis is not one of disconfirming **instances** —— for how could that be, a factor which adds to the strong evidence that we should be talking in terms of engineering not scientific practice——but from pressure concerning foundations.

The basic pressure is coming from the methodology of software engineering (SE) and its unlikely allies, and their belief that software development must proceed by a certain path: that of SPIV. Some work in expert systems, at least of the more simple minded variety, is a leading edge of this pressure on AI, because it shares the central SPIV assumption that applications are, or should be, to areas of phenomena that are completely specifiable as to their behaviors, and specifiable in advance, not during the process of programming. We shall discuss this issue in detail below; here we just want to note a key ally of SPIV, and one that might be thought historically unlikely: Chomskyan linguistics and its current phrase-structure grammar successors (e.g. Gazdar,1983).

The natural language case is a central and relevant one, for it is the area of human phenomena modelled by AI where the strongest case can be made that the data are not of a type that allows complete pre-specification, in the sense that that would be the case if the set of sentences of, say, English were a decidable set. Yet, Chomsky's intention was always to show that his grammars did cover such a set, and even though that enterprise failed his successors have made it a central feature of their claims about grammar that the set to be covered should be recursively decidable (Gazdar 1984). In the sense under discussion, therefore, some recent work in AI and natural language processing (NLP) has been an example of SPIV methodology and in an area where it is, to some at at least, the most counter-intuitive. We shall expand on this point below.

Our claim then is that AI methodology is under threat from an opposing paradigm, one not appropriate to AI's subject matter, and one that encompasses conventional SE plus much of current ES and areas of NLP. As we shall discuss below, the SPIV paradigm is not a viable one for practical software development, and it is a matter of some contention as to whether the current lack of practical utility, is a logical necessity, or just a puzzle to be eventually solved in the course of normal science or engineering.

Nevertheless, what we do see in the methodology of practical software engineering is a firm adherence to both prior specification of the problem and clear testability of program behavior, in that any given instance of program behavior is decidably correct or incorrect. We shall call this methodological variant on SPIV, which lacks the stronger requirement of the proof of the software's correctness, SAT.

The advocates of SPIV

Dijkstra (1972) laments the state of programming and predicts a revolution that will enable us "well before the seventies have run to completion ... to design and implement the kind of systems that ... will be virtually free of bugs." Two key arguments that he uses are:

(1) "the programmer only needs to consider intellectually manageable problems"

(2) "The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness ... correctness proof and program grow hand in hand."

Almost ten years on, in his preface to Gries (1981), Dijkstra says: "the 'program' we wrote ten years ago and the 'program' we can write today can both be executed by a computer, that is about all they have in common ... . The difference between the 'old program' and the 'new program' is as profound as the difference between a conjecture and a proven theorem, between pre-scientific knowledge of mathematical facts and consequences rigorously deduced from a body of postulates."

Gries himself is more cautious. He admits the truth of the charge that the formal approach to reasoning about programs has only been successfully applied to small (and we might add) abstract, problems. Nevertheless, he writes "I believe the next ten years will see it extended to and practiced on large programs." Gries sees himself as taking "a middle view" on proving program correctness: "one **should** develop a proof and program hand-in-hand, but the proof should be a mixture of formality and common sense."

Clearly this is a weakened view of the proof notion in software engineering, but still it takes for granted that there is some crucial essence of the program (i.e., the underlying algorithm) to be proven correct with respect to the problem specification, and further that this is the sort of process that admits the possibility of classical proof.

Hoare (1981) in his Turing Lecture states that "A lack of clarity in specification is one of the surest signs of a deficiency in the program it describes, and the two faults must be removed simultaneously before the project is embarked upon." Hoare also believes in the necessity of proving programs correct, and his axiomatic semantics is a formalism designed to do just that.

A "sad remark" in Dijkstra's (1976) book is: "we have witnessed the proliferation of baroque, ill-defined and, therefore, unstable software systems. Instead of working with a formal tool, which their task requires, many programmers now live in a limbo of folklore, in a vague and slippery world, in which they are never quite sure what the system will do to their programs. Under such regretful circumstances the whole notion of a correct program - let alone a program that has been proven correct - becomes void. What the proliferation of such systems has done to the morale of the computing community is more than I can describe."

These are not clear statements of what we have called SPIV, but they all take it for granted: (a) that a formal specification of the problem is a necessary prerequisite of serious software engineering; and (b) that formal proof of correctness of a program is, at least in principle if not in practice, possible. They admit no alternative in software engineering science. They hold SPIV as the single exemplar to aspire to.

We can draw two possible implications from this hardline SPIV viewpoint:

either (i) implementation of certain problems should never be attempted because they are not formally circumscribable problems; hence AI software is a fundamentally misguided notion.

or (ii) the absence of formal specifications in some areas is only a reflection of our current ignorance, and eventually problems such as NLP will yield to formal circumscription and thus the SPIV methodology. The present task is to develop the requisite formalisms first, rather than hack at implementations.

We argue that the truth of (i) is an open question but would be widely resisted by members of AAAI presumably, and that (ii) is false. Certain problems are not formally circumscribable, even in principle, but nevertheless there are possibilities quite outside the SPIV paradigm for producing robust and reliable software products. But only if this possibility is accepted can we expect sufficient work on the alternative RUDE methodology to generate a discipline of incremental program development.

Dijkstra (1972) states "an article of faith ... viz, that the only problems we can really solve in a satisfactory manner are those that admit a nicely factored solution." He further expects that living with this limitation "will repeatedly lead to the discovery that an initially intractable problem can be factored after all." Thus Dijkstra seems to favor implication (ii) above -- the one that appears to be more promising for the practical applications of AI, but also the one that we believe is quite demonstrably wrong.

Practical SE is still SPIVish

It can be argued that SPIV is just a straw man, and is just not a practical methodology. It might be said that programming realists know that iteration, incremental development, and testing for correctness are essential components of practical software development. We maintain that this is true but that the general SPIV paradigm under which they operate only permits exploration of alternatives that are consistent with the two key assumptions of SPIV given earlier. In fact, assumption (b) is weakened to correct/incorrect testability of specific program behaviors: from proof of correctness of an algorithm to clear testability of specific instances of behavior. And this is what we are calling SAT (complete Specification and Testability), and we will maintain that it is still quite distinct from RUDE.

First, we can see the crucial role of specification (and this is the key assumption that to some extent implies the clear testability of program behavior). Thus, we find this assertion in Liskov and Guttag (1986), "The principle tenet of the book is that abstraction and specification are the linchpins of any effective approach to programming."

Yourdon (1975), an important figure in practical, large-scale software engineering, makes the general point about the feasibility of top-down design (the most popular strategy) being critically reliant upon the prior existence of a complete and rigorous specification. He says: "It is extremely difficult to develop an organized top-down design from incoherent, incomplete, disorganized specifications." None of those qualities is desirable, of course, but although we can eliminate the first and third from AI, we must learn to live with the second.

>From the largely unquestioned assertion about the centrality of SAT in conventional software engineering, we can briefly examine some recent, AI-related, attempts to propose more realistic practical program development methodologies -- attempts that seem to recognize the crucial role of incremental program development in some problem areas. At first sight these "new paradigms" appear to be attempts to develop the RUDE methodology, but on closer inspection we can see that they are actually SAT proposals.

Balzer, Cheatham & Green (1983) call their scheme "a new paradigm" for "software technology in the 1990's." Essentially what they propose is that the iteration and incremental development is restricted to a formal specification which can be more or less automatically transformed into an implementation. This proposal clearly hinges on the prior and continued availability of a formal specification, and this being the case it is a SAT scheme that does not solve our problem. This approach has been suggested as a solution to the problem of incremental program development in AI (Mostow, 1985), and some of the problems with it have been pointed out elsewhere (Partridge, 1986a briefly, and at length in Partridge, 1986b).

Kowalski's (1984) presentation of a "new technology" for software design describes an iterative, trial-and-error process for "analysing the knowledge that lies behind the user requirement." But once we have a formal specification the situation is different: "Good programmers start with rigid, or at least formal, software specifications and then implement them correctly first time round — never get it wrong." Clearly formal specification is still the key assumption, and the notion of a correct implementation is there also. This is a scheme that is at least SAT, if not something a lot closer to true SPIV.

Rapid prototyping is a key idea in the methodology of expert systems development: build a quick, small-scale version in order to generate an understanding of the problem such that the real system can then be specified and implemented. We accept the value of exploring the application domain using a working program, but the implication is that one preliminary venture into the field will be sufficient to support a full-scale specification of the problem. It is almost as if one iteration of SPIV followed by SPIV is expected to take care of the performance-mode specification aspects of AI problems; and this seems unlikely to us.

Much more akin to the spirit of the RUDE methodology are some of the general schemes that are being abstracted from the practice of constructing expert systems.

The "stages in the evolution of an expert system" described by Hayes-Roth, Waterman & Lenat (1983) are:

IDENTIFICATION - determining problem characteristics

CONCEPTUALIZATION - finding concepts to represent knowledge

FORMALIZATION - designing structures to organize knowledge

IMPLEMENTATION - formulating rules that embody knowledge

TESTING - validating rules that embody knowledge

Clearly, there is still a belief in the desirability of formalization, and this is quite consistent with RUDE, for it is not a complete a priori formalization of the problem. Testing is also a feature of the above design scheme, but it is not the testability of SAT: in general it is a judgment of adequacy of system output. We begin to see a significant departure from the SPIV paradigm, and this will eventually lead us into the RUDE paradigm.

Circumscribability and decidability as needed for SPIV

Let us then stand back and look at why SPIV cannot be applied to certain areas of phenomena. That applicability requires both:

a) circumscribability of behavior, in that the data must form a recursive, decidable set

b) in its strongest form SPIV requires openness to proofs of the program.

Natural language at least is the clearest example of a phenomenon where this is not possible: the set of meaningful sentences, however described, will be confronted with meaningful utterances outside it. We could all perform this operation if called upon to do so -- it is not so much a matter of ingenuity as part of the processes of everyday life. This case is set out in detail in (Wilks 1971). If that case is correct, all current post-Chomskyan attempts to introduce what is essentially SPIV methodology into AI/NLP drag that part of AI in the direction of the SPIV/SAT methodology, are misguided, and should be abandoned. Natural language will be understood by machines in terms of complex pattern-matching and motivated relaxation of rules, or the accommodation of new data to existing representations.

The second requirement, involving the nature of proof, is more complex, and applies only to the stronger forms of SPIV. This again can be reduced to proof by example: there just are no useful striking examples of basic AI programs proved correct, ones where trust in them in any way depends on that proof. We agree with De Millo et al (1979) that the history of mathematics suggests that program proof could never be more reliable in principle than proof in mathematics, and that has shown itself to be a shifting ideal, utterly dependent on time-dependent social standards.

The conclusion from this, for us, is not at all the advocacy of a new methodology for AI but a call to return to RUDE, which is the classic methodology of AI, and its distinctive feature in both areas (of behaviors and proof) is that traditionally associated with the term "heuristic": that which does not admit of formal proof. What is needed is proper foundations for RUDE, and not a drift towards a neighboring paradigm.

An introduction to RUDE methodology

If all problems are not subsumable under SPIV or SAT, and we do not wish to abandon the possibility of implementing those problems, what methodology could we use? Can we forgo the comfort of complete, prior, formal specification, the notion of program correctness, and even the clear testability of program behavior, and yet still generate useable software? It seems to us that the (almost) unanimous response to this question from CS and AI researchers alike is, NO. We believe that this negative position is both premature and entails severe limitations on the future of AI software if true. Thus we would like to see a thorough exploration of alternatives which reject the key assumptions of SPIV and SAT, and in this regard we shall offer some remarks on what the necessary methodology might look like. Essentially what we shall propose is a disciplined development of the 'hacking' methodology of classical AI. We believe that the basic idea is correct but that the paradigm is in need of substantial development before it will yield robust and reliable AI software.

It would seem that any development of RUDE must yield programs that are inferior to those of pure SPIV (with its guarantees of correctness), if only SPIV can be applied to the problems of AI. So one of the prerequisites for a serious consideration of RUDE must be to demonstrate that SPIV is an impossible ideal. SAT adherents have regretfully waived the requirement of proof. They work with SAT as an inferior stopgap that future puzzle-solving will transform into something more closely approaching SPIV; the RUDE methodology has no such pretensions.

We propose that adequate implementations of, say, the NLP problem can be generated by incremental development of a machine-executable specification. In place of the correctness notion of SPIV or the clear testability of SAT we argue for a notion of adequacy. Intelligence is not typically associated with the notion of correctness in some absolute sense. The criteria of intelligence are adequacy and flexibility. It is misguided to impose absolute binary decisions on say, the meaning of a sentence, or even on its grammaticality. More realistically there is likely to be a set of more or less adequate meanings given certain contextual constraints.

Rather than implementation of an abstract specification, we propose exploration of the problem space in a quest for an adequate approximation to the NLP problem. The key developments that are needed are methodological constituents that can guide the exploration -- since for a random search is unlikely to succeed. We can list some of the puzzles (we would claim) of the RUDE paradigm -- puzzles which, if they are indeed solvable as such, are en route to a disciplined version of the RUDE methodology.

DECOMPILING -- deriving consequences for the 'form' from observations of the 'function';

STEPWISE ABSTRACTION -- a sequence of decompiling operations;

STRUCTURED GROWTH -- techniques for reversing the usual entropy increase that accompanies incremental development;

ADEQUACY VALIDATION -- adequacy usefully considered as a lack of major performance inadequacies, etc.;

CONTROLLED MODIFICATION -- a strategy of incremental change through analysis of program abstractions, subsuming both decompiling and structured growth.

A RUDE-based methodology that also yields programs with the desiderata of practical software -- reliability, robustness, comprehensibility, and hence maintainability -- is not close at hand. But if the alternative to developing such a methodology is the nonexistence of AI software then the search is well motivated.

Further sociological complexity

Let us expand for a moment on this notion of the paradigms SPIV and SAT that are neighbors to RUDE. An illustrative table might be the following ordered list of methodologies:

1. Only properly proved programs are OK.
2. Only programs conforming to the standards of SE are OK.
3. Only programs founded upon adequate logical or linguistic theories are OK.
4. Expert systems as a form of SE are OK.
5. AI programs that exist as practical/commercial software are OK.
6. Working AI demonstration programs of the standard type in AAAI papers are OK.

OK is a hopelessly weak term here, and readers may prefer to substitute "acceptable" or "intellectually defensible" or any term they prefer. The point of the table is its order and not the predicates attached to lines. For each level, its adherents believe that those below it are NOT OK! Readers may also enjoy the exercise of attaching names of individuals or companies to each line: it is easily done, and provides an easy check on the table via the transitivity-of-scorn rule. The upwards direction in the table is not simply interpretable, but a close approximation is: anything above this line cannot be seriously performed, but might be OK if it were. By our classification 5 & 6 are RUDE, 2, 3 & 4 are SAT, and 1 is SPIV.

Putting RUDE on a better intellectual foundation

Where should we seek for this, for it must not be simply a matter of sociological issues but intellectual ones? It is a familiar argument that computer projects, particularly large scale ones (see Bennett 1982, or the classic, Brooks 1975) fail for an extraordinary range of social and organizational reasons, and that would continue to be the case even if, per impossible, realistic proofs of programs were to become available. Conversely, the problems that AI seems to have in getting out into the world, in convincing itself and others that there is or has been at least one piece of real red-blooded useful and workable AI, is not a matter of social constraints and inhibitions as Schank seemed to claim (in Schank 1983). The fact is that the only saleable parts of real AI at the moment (apart from chess games, perhaps) seem to be toolkits for building other bits (which might seem to confirm Bundy's view, expressed in his reply to Schank (Bundy 1983) that AI is really a toolkit set, but not in a way that he would like).

Even if RUDE is, in some sense, the basic, classical, method of advance in AI, would it be sufficient to simply declare that and carry on as before? Almost certainly not, and even if AI were deemed to be at some level engineering, there too the search for proper defensible foundations cannot be avoided.

One natural place to look is for a claim that programs themselves can be theories: this has been defended before (e.g., Wilks 1974);
declared to be not impossible (e.g., Simon 1979); supported, indirectly at least, but in terms that no AI practitioner could accept (e.g., Sampson 1985, which requires that the area under study be a closed form like that of a dead language—this would be a drastic return to the very Chomskyan assumptions that RUDE advocates would reject); and even demonstrated by example (Partridge, Johnston, and Lopez, 1984).

Note that it might be a reductio ad absurdum to have programs as theories, but this is just the kind of reductio that AI has willingly embraced in the past: consider, for example, the position that seems natural to many AI researchers in which every truth in a world is considered an axiom, a reductio Tarski foresaw.

Finally, we note that these issues of methodological validity may not be a purely parochial concern of the AI community. It has been suggested (e.g., Giddings 1984) that the SPIV/SAT paradigm may be inappropriate for much of SE as well. It may be that RUDE should be the *major* paradigm instead of SPIV/SAT.

References

Balzer, R., Cheatham, T. E., & Green, C. 1983. Software Technology in the 1990's: Using a New Paradigm, IEEE Computer, Nov., pp. 39-45.

Bennett, J. 1982. Large Computer Project Problems and their Causes, Tech. Report. 188, Basser Dept. of CS, University of Sydney.

Brooks, F. P. 1975. The Mythical Man-Month, Addison-Wesley: Reading, Mass.

Bundy, A. 1983. The Nature of AI: A Reply to Schank, The AI Magazine, Winter, pp. 29-31.

De Millo, R. A., Lipton, R. J., & Perlis, A. J. 1979. Social Processes and Proofs of Theorems and Programs, Comm. ACM., 22, pp.271-280.

Dijkstra, E. W. 1972. The humble programmer, Comm. ACM, 15, 10, pp. 859-866.

Dijkstra, E. W. 1976. A Discipline of Programming, Prentice-Hall: Englewood Cliffs, NJ.

Duffy, M. C. 1984. Technomorphology, Engineering Design and Technological Method, Proc. Annual Conference of the Brit., Soc. for the Philosophy of Science.

Gazdar, G. 1983. NLs, CFLs, and CF-PSGs. In Sparck-Jones and Wilks (eds.) Automatic Natural Language Parsing, Ellis Horwood: West Sussex, UK.

Giddings, R. V. 1984. Accommodating Uncertainty in Software Design, Comm. ACM, 27, 5, pp. 428-435.

Gries, D. 1981. The Science of Programming, Springer-Verlag: NY.

Hayes-Roth, F., Waterman, D. A., & Lenat, D. B. 1983. Building Expert Systems, Addison-Wesley: Reading, Mass.

Hoare, C. A. R. 1981. The Emperor's Old Clothes, Comm. ACM, 24, 2, pp. 75-83.

Kowalski, R. 1984. Software Engineering and Artificial Intelligence in New generation Computing, The SPL-Insight 1983/84 Award Lecture.

Kuhn, T. S. 1962. The Structure of Scientific Revolutions, University of Chicago: Ill.

Liskov, B. & Guttag, J. 1986. Abstraction and Specification in Program Development, McGraw-Hill: NY.

Mostow, J. 1985. Response to Derek Partridge, The AI Magazine, 6, 3, pp. 51-52.

Partridge, D. 1986a. RUDE vs COURTEOUS, The AI Magazine, 6, 4, pp. 28-29.

Partridge, D. 1986b. Artificial Intelligence: applications in the future of software engineering, Ellis Horwood & Wiley: UK.

Partridge, D., Johnston, V. S., & Lopez, P. D., 1984 Computer Programs as Theories in Biology, J. Theor. Biol., 108, 539-564.

Schank, R.C. 1983.The Current State of AI: One Man's Opinion. AI Magazine, Winter/Spring, pp. 3-8.

Sampson, G. 1975. Theory choice in a two-level science. Brit. Jnl. Philos. of Science. pp. 97-107.

Simon, T.W. 1982. Philosophical Objections to Programs as Theories. In. Ringle, Philosophical Perspectives in AI, Humanities Press: Atlantic Highlands, NJ.

Wilks, Y. 1971. Decidability and Natural Language. Mind. 80. pp.497-516.

Wilks, Y. 1974. One Small Head: Models and Theories in Linguistics. Foundations of Language. 11. pp.77-95.

Wilks, Y. 1986. Bad Metaphors: Chomsky and Artificial Intelligence, In S. & C. Mogdil (eds) Noam Chomsky: Consensus and Controversy.

Yourdon, E. 1975. Techniques of Program Structure and Design, Prentice-Hall: Englewood Cliffs, NJ.