Problem description and hypotheses testing in Artificial Intelligence[1]

Paul Mc Kevitt and Derek Partridge[2]

ABSTRACT

There are two central problems concerning the methodology and foundations of Artificial Intelligence (AI). One is to find a technique for defining problems in AI. The other is to find a technique for testing hypotheses in AI. There are, as of now, no solutions to these two problems. The former problem has been neglected because researchers have found it difficult to define AI problems in a traditional manner. The second problem has not been tackled seriously, with the result that supposedly competing AI hypotheses are typically non-comparable. If we are to argue that our AI hypotheses do have merit, we must be shown to have tested them in a scientific manner. The problems, and why they are particularly difficult for AI are discussed. We provide a software engineering methodology called EDIT (Experiment Design Implement Test) which may help in solving both problems.

1 INTRODUCTION

One of the most serious problems in Artificial Intelligence (AI) research is describing and circumscribing AI problems. Unlike traditional computer science problems, problems in AI are very hard to specify, never mind solve (see [1], p. 53-87). Take, for example, the problem of natural language processing in AI. The problem of building a program, P, to understand any natural language utterance which is passed to P is very difficult. There will always be some word, or utterance structure, or relationship between utterances, which the program will not understand. In any event, it is hard to circumscribe the problem, as there are so many words and sentence structures, and it would take a long time to specify all of these. This point is elaborated with more detail by Mc Kevitt in [2]. Also, Wilks in [3] (p. 131) says, "However, if there is any validity at all in what I called the circumscription argument, then this is not so, for a natural language cannot be viewed usefully as a set of sentences in any sense of those words. The reason for this, stated briefly and without detailed treatment given in [4] and [5], is that for no sequence of words can we know that it cannot be included in the supposed set of meaningful sentences that make up a natural language."

[1]This research has been funded in part by **U S** *WEST* **Advanced Technologies**, Denver, Colorado, under their Sponsored Research Program.

[2]The authors' address is Department of Computer Science, University of Exeter, GB Exeter EX4 4PT, EC. E-mail: {pmc,derek}@cs.exeter.ac.uk

However, it is possible to limit the problem of natural language processing by taking a specific domain and have a system which only accepts utterances within a prescribed domain. Take the problem of building a natural language interface which answers questions about the UNIX operating system. Such a program, called OSCON, is proposed by Mc Kevitt in [6]. Yet, even in this simplified case the difficulty of describing the problem, and testing solutions to it, remains. It is hard to define what sort of questions people will ask as there are so many of them. AI researchers tend to try and predict what sort of questions people "might" ask and develop a system to cater for these. The question we would like to ask here is, "Is there any better method of defining AI problems rather than just, a priori, deciding on a few of the specific components of the problem?"

A second problem arises out of the first, when we do not limit our goal to a few prespecified examples, and that is how to test a solution to an AI problem if such a potential solution is found. Today, the test of solutions to AI problems is the implementability test; i.e. if a program can be implemented and demonstrated to solve some selected examples of the problem then that program is considered a good one. This is, of course, no test at all. If AI is a science then there needs to be some method for testing or experimenting with programmed solutions rather than just implementing and demonstrating them. We take a quote from Partridge in [1] (Preface) which emphasizes the point, "AI is a behavioral science; it is based on the behaviour of programs. But it has not yet come to grips with the complexity of this medium in a way that can effectively support criticism, discussion, and rational argument - the requisites of scientific 'progress' are largely missing. Argument there is, to be sure, but it is all too often emotionally driven because rational bases are hard to find." This leads us to ask the question, "Is there any better method than just implementation and demonstration which tells us if an AI hypothesis or theory is correct?

We believe that the questions we ask here are central to the methodology and foundations of AI, and answers to the questions need to be found. Otherwise, there will be little disciplined progress in the development of AI systems and the science of AI.

2 DEVELOPING AI PROGRAMS

Computer science has already developed techniques for describing problems. Programs have been developed to aid programmers in developing and testing software for specific domains. Practical software engineering tools such as CASE (Computer Assisted Software Engineering) products are being used today (see [7]). CASE tools are very useful for building computer software for limited domains, yet not very useful for tackling problems outside their scope. For example, there are very few CASE tools which would be useful for tackling the problem of building a machine translation system for translating Swahili into Chinese.

Methodologies for the development of computer software have also been defined. Partridge in [1] (p. 91-96) argues for the a Run-Debug-Edit methodology which is

later modified to RUDE (Run-Understand-Debug-Edit) in [8]. RUDE is a software development methodology for the design of, mainly, AI programs. It is also pointed out by Partridge in [1] that the RUDE methodology may be useful for traditional computer science problems too. This methodology calls for a discipline of incremental program development where programs are run and, if they fail on input, are edited and rerun. Partridge and Wilks in [8] (p. 369) say, "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 the paradigm is in need of substantial development before it will yield robust and reliable AI software." The problem with RUDE is that it is tedious, and takes a long time, as the programmer is just hacking piecemeal at solving the problem without really knowing what the problem is. There is no specific goal toward which the program or programmer converges.

Another methodology called SAV (Specify-And-Verify), coined by Partridge in [9], calls for formal specification of problems, and formal verification of the subsequent algorithm. The SAV approach is advocated by Dijkstra, Gries, and Hoare in [10], [11], and [12] respectively. The use of formal techniques in proving programs correct for real world complex problems in computer science has proven difficult. One of the problems with Artificial Intelligence (AI) programs is that, as we've said, they are very difficult to specify. The application of proof logics to the intricacies of complex programs is too tedious and too complex. The technique has only become useful for small, simple programs.

Both RUDE and SAV only ensure that a program is developed for a particular specification. There is no guarantee that the specification is correct or solves the real world problem at hand for which it is intended.

3 THE EDIT METHODOLOGY

EDIT (Experiment-Design-Implement-Test) is a software development methodology which attempts to integrate elements of the SAV and RUDE methodologies. EDIT incorporates experimentation as an integral component. This is particularly useful in the AI problem domain which incorporates the added difficulty of the researcher not knowing how to describe a problem while trying to solve it. Briefly, EDIT has the following stages:

1. **Experiment:** Experiment(s) (E) are conducted to collect empirical data on the problem. This data can be stored in log file(s) (L).

2. **Design:** A design[3] (D), or specification, can be developed from L together with relevant theories of the program domain.

3. **Implement:** The description, or specification, is implemented (I) as a computer program (P).

[3]By "design" we mean any reasonable description whether it be in English, Hindi, Gaelic, logic, algorithmic form, or assembly code.

4. **Test:** P is sent around the cycle and tested by placing it through E again. However, this time E involves P whereas initially E did not involve a program. The cycle is iterated until a satisfactory P is found.

The system developer(s) initially use(s) E to help define the problem, and successively use(s) E to develop and test P. In the initial stage E does not involve a program. However, each subsequent E involves a partially implemented P until the final P is decided upon. EDIT will always terminate after E and before I in the cycle. The EDIT cycle is shown in Figure 1 below.

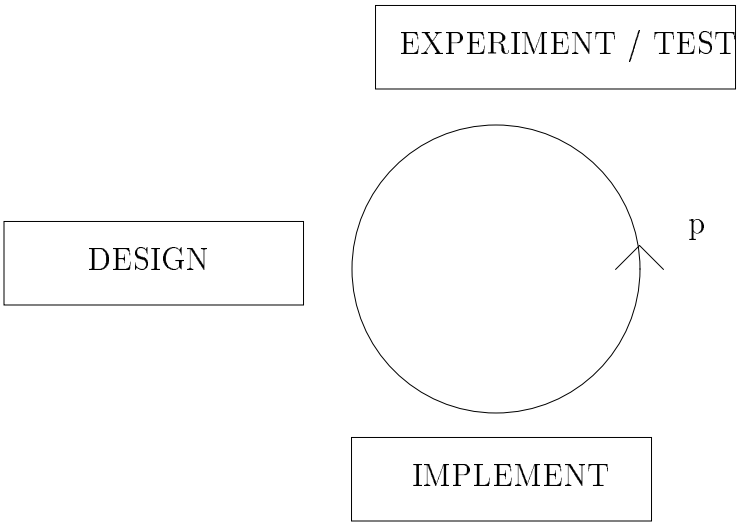

Figure 1: EDIT (Experiment Design Implement Test) Cycle.

At the experiment stage (E) an experiment is conducted to gather data on the problem or the quality of the current potential solution, P_n. Say, for example, the problem is to develop a natural language program which answers questions about computer operating systems like UNIX. Then, valid experimentation software would be a program which enables a number of subjects (S) to ask questions about UNIX, and an expert to answer these questions. An example setup for this experiment would be the Wizard-of-Oz[4] paradigm. In fact, an augmented Wizard-of-Oz technique is used where a program (P) can be inserted and used in the interaction with the subject. A number of S and Wizard(s) (W) from varying backgrounds may be used in the experiment. Of course, the greater the number of S and W the more

[4] A Wizard-of-Oz experiment is one where subjects interact with a computer through typed dialogue at a monitor and are led to believe that they are conversing with the computer. For example, in the case of a Wizard-of-Oz test for a natural language interface, a subject's utterances are sent to another monitor where a "Wizard", or expert, sends back a reply to the subject monitor.

comprehensive the data collected will be. Also, there may be groups of S and W rather than just a single S and single W. Information on exchanges between S and W is logged in a log file (L) for later inspection. S and W operations are flagged in the file. Such an experiment is described with greater detail by Mc Kevitt and Ogden in [13] and the implications of that experiment are described by Mc Kevitt in [2].

At the design stage (D), L from E is analysed and inspected. In the initial stage the data here gives a snapshot description of the problem and how it is characterised. Further stages of the cycle will give snapshots of how well the problem is characterised in the current P. An analysis of L will give a picture of the information needed in various components of a program, such as knowledge representation, user modeling, and reasoning components. Many researchers and domain experts from various backgrounds may be called in to analyse L and determine what aspects of the software need to be developed. In fact, the type of researcher brought in will determine the type of program eventually developed and the best of all worlds would be to have a wide span of researchers/experts from different backgrounds. The job of the researchers is to develop algorithms with the help of the data and to specify these algorithms in some manner.

At the implementation (I) stage the algorithms or designs in D are implemented. These designs may be implemented in any programming language (P) that the implementers find most appropriate. Finally, the implemented program is sent back to E again and tested. Then, a new cycle begins.

During the initial state of the EDIT manifestation described here, S and W interact over the problem and the data is logged in L. Data may be collected for a specific task within a domain, or the whole domain itself. Each successive run of E involves the incorporation of P, which tries to answer questions first and if it fails W steps in while P restarts. The cycle may be operated in real time, or batch mode. In batch mode the experimentation component would involve a number of batched questions which are collected from S, and processed by P, with W interrupting where P fails.

The EDIT cycle continues until the program performs satisfactorily to the requirements of the designers. The designer(s) may wish P to perform satisfactorily only 50% of the time, or 80% of the time. The success or failure of P will be determined at design time, D, when the L is analysed. L will show where P has failed and where it has passed the test. W entries will show up why P did not work and will indicate what components of P need to be updated. In the case of natural language question answering W entries might show up the fact that certain types of question are not being answered very well, or at all. Therefore, W entries would indicate how P needs to be augmented in principle to solve a recurring pattern of failure. W entries could be analysed for such recurring patterns. In effect, what is happening here is that P is "learning" by being investigated and augmented in the same way as a mother might teach her child noticing the child's failure to complete

certain tasks[5].

The success of P is measured by the number and type of answers P can give, and the number of answers P gets correct. The measure of capability and correctness is determined by inspection of L. During the development of P the initial coding may need to be recoded in some manner as data collected later may affect P's design.

There are many forms in which the EDIT cycle may be manifested. The experimentation stage may involve experiments other than the Wizard-of-Oz type. Another experiment might involve an observer sitting beside the subject during testing and helping the subject with the program as he/she uses it and also restarting the program itself.

The design stage and inspection of L may involve only one, or a number of designers. These designers may know much about the domain, and little about design, or vice versa. Experts and good designers may both be used at the design stage. Also, E could consist of a set of experts with different points of view and different backgrounds. E is in the spirit of Partridge and Wilks in [8] (p. 370), "Rather than the implementation of an abstract specification, we propose exploration of the problem space in a quest for an adequate approximation to the NLP problem." Hence, EDIT may consist of many manifestations of the methodology, yet, the basic methodology involves developing and testing through experimentation.

EDIT is a useful technique in that it allows the iterative development of systems and gives feedback on how to design an AI system as it is being developed. Sharkey and Brown in [14] (p. 282) point out that the belief that an AI system can be constructed first, and then tested later, as argued by Mc Kevitt in [2], is not the way to go. Sharkey and Brown show that (1) an AI system takes a long time to build, and it may be wrong at the beginning, and (2) an AI theory, and its implementation in the final state, may not be configured in a way that allows psychological testing.

It is important to point out here that the idea of using a Wizard for testing AI programs has a parallel in standard software engineering (see, for example, [15]). In the testing of standard software, top-down testing schemes use dummy modules or "program stubbs". The modules can be implemented with the following constraints: (1) Exit immediately if the function to be performed is not critical, (2) Provide a constant output, (3) Provide a random output, (4) Print a debugging message so that the programmer knows the module is entered, (5) Provide a primitive version of the final form of the module. Yourdon in [15] points out that in theory top-down testing could be done with only the main program and with all the lower-level modules implemented as stubs. However, he notes that in practice this would be clumsy. The same holds true for the Wizard-of-Oz experiment: an initial analysis of data might be one where most of the answers are given by the Wizard and only a few are handled by the system. This would be clumsy in practice as the Wizard would end up answering most of the time and the system would only respond to a few utterances here and there.

[5]This analogy was provided in personal communication by Brendan Nolan of University College Dublin (UCD).

Pressman in [16] (p. 508) discusses the use of stubs and points out that subordinate stubs can be replaced one at a time with actual modules. Tests are conducted as each module is integrated. On the completion of each test another stub is replaced with a real module. Also regression testing may be conducted where all, or some, of the previous tests are rerun to ensure that new errors have not been introduced.

4 THE SCIENCE OF AI

Now that we have described our position on a methodology for developing good algorithms we shall move on to the problem of testing them. There needs to be some technique for testing if the algorithm works. Today, the test of AI theories is one where the programs, embodying those theories[6], are implemented, and demonstrated to work, over a few selected examples. This, however, is not a test at all, as any AI theory, or hypothesis, can be implemented. At most, researchers tackle the problem of testing AI systems in a weak sense by showing that they work for a few examples.

One of the problems with AI today is that it is not appreciated as a science and has no scientific test methodology. Narayanan in [19] (p. 46-47) points out, "It can be argued that the criterion of implementability is vacuous at the level of the Church-Turing thesis". The thesis basically says that any process which can be described by an algorithm can be implemented on a computer. Thus, any AI theory which can be described by an algorithm can be implemented on a computer, and hence all AI theories are valid no matter what they say. Sharkey and Brown in [14] (p. 278) also point out this problem: "To say that a theory is implementable is simply to say that it can be expressed in the form of a computer program which will run successfully", and suggest that a solution needs to be found (p. 280), "Another question we would like to raise here is this: At what point in implementation do we decide that there are too many patches to accept that the running program is actually a test of a theory." Sutcliffe in [20] argues for more empiricism and says, "I see the use of norming studies and other techniques from psychology as being relevant to AI." EDIT calls for not just implementability but also for the implementation to work on experimentation over real data. Also, EDIT moves forward on helping to solve the problem of how to check whether an AI theory is valid. Narayanan in [19] (p. 48) points out, "In any case, even if a criterion of complexity for AI programs (theories) can be found, there still remains the suspicion that no criterion exists for determining whether an AI theory is true or accurate." EDIT provides a criterion for the testing of theories embodied in programs by the inspection of log files.

There have been many arguments as to whether AI is, or is not, a science (see [21], [22]). Schank in [23] brings up the question as to whether AI is a technology of applications or a science. He points out that researchers have taken two directions,

[6]We do not make any strong claims here as to the relationship between programs and theories. However, this issue is discussed in [17], [18], and [3].

the scientists interested in working on problems like the brain or more neat logic problems, and the applications people working on building real practical systems.

Bundy in [24] calls AI an "engineering science". He says that it consists of the development of computational techniques and the discovery of their properties. He argues that AI is exploratory programming where one chooses a task that has not been modelled before and writes a program to implement it. On the other hand Dietrich in [25] (p. 224) argues that AI is a science and says, "Then, I will suggest a new theoretical foundation, and argue that adopting it would provide a clear, unequivocal role for programs: they would be controlled experiments, and AI would become a science." He points out that such experiments can be operated over natural systems such as ecosystems, and populations such as ant colonies. He says (p. 231), "In the science of intelligent systems, therefore, computer programs would have a definite role: they would allow scientists to experiment with hypotheses about the nature of intelligence".

Sparck Jones in [26] (p. 274) says that AI is engineering and points out that "... AI experiments are engineering experiments serving the designs of task systems, i.e. of artefacts." However, although we would agree with Sparck Jones in the sense that AI programs can be tested and redesigned by such experiments we would argue that AI hypotheses can also be tested with experiments such as those argued for in EDIT. Such experiments might be scientific ones, rather than engineering ones.

Let's assume that AI is a science. One of the problems is then to decide what the methodology of this science is. Narayanan in [21] (p. 164) brings up the point nicely, "The relationship between AI and cognitive psychology is strong. Does that mean that AI theories must conform to the same methodological rigour as psychological theories? If not, then a clear methodology must be provided for constructing and testing AI theories, otherwise AI might end up being a completely speculative subject, more akin to science fiction than science." We must also ask the question of how to test hypotheses in AI if it is a science.

EDIT acts as methodology for testing hypotheses in AI where such hypotheses may be solutions to parts of problems. The advantage of the Wizard-of-Oz technique incorporating W, is that if P fails for reasons other than the hypothesis then the W can step in, keeping P alive, so to speak. Meanwhile, no data is lost in the current experiment dialogue. Of course, the log file marks where W interrupts. During testing as far as S is concerned the program has never failed as S does not necessarily know that W has intervened. The data from the testing phase can be logged in a file and system developers can then observe where the system failed, and where the W interfered. This information will be used in updating the system and any theory which the system represents.

EDIT addresses the problem brought out by Narayanan in [19] (p. 44) where he says, "The aim of this paper, apart from trying to steer well clear of terminological issues, such as the distinction between 'science' and 'study', is to demonstrate that unless AI is provided with a proper theoretical basis and an appropriate methodology, one can say just about anything one wants to about intelligence and not be contradicted; unless AI is provided with some reasonable goals and objectives little

of current AI research can be said to be progressing." It is believed that EDIT might be the methodology that Narayanan asks for.

We would like to point out that EDIT is compatible with the methodologies of three philosophies of science: (1) the narrow inductivist, (2) the Hempel approach, and (3) the Popper approach. The 'narrow inductivist conception of scientific inquiry' (see [27]) is one which follows: (a) observation and reasoning of all the facts, (b) analysis and clarification of these facts, (c) inductive derivation of generalisations from them, and (d) further testing of the generalisations. This is in accord with EDIT when used in the sequence, Experiment-Design-Implement where log files are observed, and a program is developed from them.

However, Hempel in [27] argues that this type of scientific inquiry is not useful, and his approach is to develop hypotheses as tentative answers to a problem under study, and then subject them to empirical test. Hempel argues that the hypothesis must be testable empirically and that even if implications of the hypothesis are false under testing, the hypothesis can still be considered true. Hypotheses can be modified under experimentation until a limit is reached whereby the theory has become too complex and a simpler theory should be sought. Again, EDIT is compatible with Hempel's test ([27]) of AI systems as scientific hypotheses. EDIT can allow the AI scientist to have an hypothesis, a priori, (D) and place it into the cycle at I where it will be passed to E. If the program P fails at E then the log file must be analysed to see why it failed. If the hypothesis has failed then P can be modified and tested again.

Popper in [28] argues that scientific hypotheses must be developed and if they fail a scientific test then they must be thrown away and a new hypothesis developed. There is no room for hypothesis modification. Also, there must be a test which can show the hypothesis to be false. EDIT can allow scientific testing in the Popperian framework where again an hypothesis is formulated at D, is implemented at I, and is then tested at E. If the hypothesis fails then a new one must be developed and placed into the cycle at D again.

Marr in [29] describes two types of theory. The first type (type I) of theory is one where one uses some technique to describe the problem under analysis. Marr refers to Chomsky's notion of "competence" theory for English syntax as following this approach. The point is that one should describe a problem before devising algorithms to solve the problem. The second type (type II) of theory is one where a problem is described through the interaction of a large number of processes. He points out that the problem for AI is that it is hard to find a description in terms of a type I theory. Most AI programs have been type II theories. The EDIT technique enables the development of both types of theory in Marr's terms. A type I theory can be developed in terms of developing an initial complete description (starting at D) and then implementing it (at I) and testing it (at E). Also, a type II theory can be developed by starting at E stage and iteratively developing the description D of the complete complex process.

Marr in [30] defines a three-level framework within which any machine carrying out an information processing task is to be understood:

Computational theory (Level 1): The goal of the computation and the logic of how it can be carried out.

Representation and algorithm (Level 2): The implementation of the computational theory and the representation for the input and output. Level 2 also involves the algorithm for the transformation.

Hardware implementation (Level 3): The physical realisation of the representation and algorithm.

EDIT can be described in terms of Marr's framework where level 1 is the level at which D is completed, although D does not necessarily ask for a logic. Level 2 is also conducted at D. Level 3 is conducted at the I stage. Marr does not discuss experimentation in his three level framework.

EDIT is an attempt to address the problem brought forth by Narayanan in [21] (p. 179) where he says, "What we need here is a clear categorization of which edits lead to 'theory edits', as opposed to being program edits only. It is currently not clear in the AI literature, how such a categorization might be achieved. AI does not have the sort of complexity measure which would help identify when the theory, as opposed to the program, should be jettisoned in favour of another theory." Using EDIT an inspection of L should show up, in many cases, where a program has failed because of an hypothesis failure, or because of other reasons, and hence there will be distinct implications for the theory and the program. Also, Narayanan in [21] (p. 181) says, "But given the above comments, it appears that there can, currently at least be no scientific claims for claiming that one AI theory is better than another and that AI is making progress, simply because the conceptual tools for measuring one theory against another, and so for measuring the progress of AI are missing." We believe that EDIT may be a step along the road to such conceptual tools. It may be the case that EDIT has a lot to say in the development of foundations for AI as a science rather than a technology (see [21], [22]).

5 COMPARING EDIT TO RUDE

EDIT is not just a rearrangement and renaming of RUDE. The difference is that EDIT offers a means of convergence on a solution. EDIT is a significant refinement which we expect will be widely (although not universally) applicable in AI. The difference between EDIT and RUDE is that the algorithms are developed in conjunction with data describing the problem rather than from what the problem "might" be. Too often in the field of AI there are attempts at deciding, a priori, what a problem is without any attempt to analyse the problem in depth. As was pointed out earlier one of the problems with developing AI programs is that it is very difficult to specify the problem. One solution to that might be to collect data on the problem, rather than algorithms being concocted from hopes, wishes and intuition. The second major difference is that experimentation involves testing software over real data in the domain. Also, by using the Wizard-of-Oz technique

the testing phase breaks down less as the wizard keeps the system going. We argue that this is important because if a test fails then data can be lost due to temporal continuity effects. Failure happens a lot while testing AI programs. For example, if one is testing a natural language interface, with an hypothesis for solving reference in natural language dialogue, then if the test fails the continuation of that dialogue may never happen, and data will be lost.

The problem with RUDE is that it does not include any goal as part of the process of development; only the update of a program. We argue here that E must be included to produce log files which measure how close P is to the goal that needs to be achieved. EDIT can be considered a more "tied down" version of RUDE where it is clearer what the problem is, and how well P is solving the problem. In fact Partridge and Wilks in [8] (p. 370) say, "What is needed are proper foundations for RUDE, and not a drift towards a neighbouring paradigm." Also, Partridge and Wilks in [8] (p. 370) point out a recognition of the need for convergence, "The key developments that are needed are methodological constituents that can guide the exploration — since a random search is unlikely to succeed."

The EDIT cycle is conducted until the implementation performs satisfactorily over a number of tests. The EDIT cycle enables the iterative development of a system through using the problem description itself as part of the solution process. EDIT is not just an hypothesis test method, but is also a method by which the *reason* for failure of software is logged and a method where that reason does not cause data loss. EDIT is useful for the development of software in an evolutionary way and is similar to those techniques described in [31]. Again, 100% reliability is very difficult to guarantee but we believe that problem description and implementation through experimentation will lead to better implementations than either RUDE or SPIV on their own.

EDIT is like the general methodology schemes proposed by researchers who are developing expert systems. The stages for the proper evolution of an expert system are described by Hayes-Roth et al. in [32]:

- 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

This is in the spirit of EDIT where, of course, identification is similar to E and conceptualization and formalization to D, and implementation to I. However, with EDIT, E is involved in both identification and testing and we argue that this is the way to go about testing if P is to meet the problem head on.

EDIT is currently being used in the development of AI software which answers natural language questions about computer operating systems. An initial computer

program was developed called OSCON (see [33], [6], [34], and [35]) which answers simple English questions about computer operating systems. To enhance this research it was decided that an experiment should be conducted to discover the types of queries that people actually ask. An experiment has been conducted to acquire data on the problem. More details on the experiment and its implications are given in [2].

There are probably AI domains where EDIT will fit nicely and other domains which will not — i.e. are not open to simple data collection. For example, theories of knowledge representation could only be developed with rather indirect inferences from data collection. We do not wish to stress that EDIT will be used for all AI domains but that it may be useful in some.

6 CONCLUSION

It is pointed out here that the EDI methodology can provide a solution to the development and testing of programs in Artificial Intelligence (AI), a field where there are no sound foundations yet for either development, or testing. EDIT is compatible with scientific test philosophies at each end of a scale, and if AI is to be a science, then a technique like EDIT needs to be used to test scientific hypotheses. A sound methodology will reduce problems of how to compare results in the field.

EDIT may help in the endeavour of transforming AI from an ad-hoc endeavour to a more well-formed science. EDIT provides a methodology whereby AI can be used to develop programs in different domains and experts from those domains can be incorporated within the design and testing of such programs.

We leave you with a quote from Partridge and Wilks in [8] (p. 370), "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." EDIT is part of such a search.

7 ACKNOWLEDGEMENTS

We would like to thank Simon Morgan, Richard Sutcliffe and Ajit Narayanan of the Computer Science Department at the University of Exeter and Brendan Nolan from University College Dublin for providing comments on this work.

8 REFERENCES

1. Partridge, Derek. *Artificial Intelligence: applications in the future of software engineering.* Halsted Press, Chichester: Ellis Horwood Limited, 1986.
2. Mc Kevitt, Paul. *Data acquisition for natural language interfaces.* Memoranda in Computer and Cognitive Science, MCCS-90-178, Computing Research Laboratory,

Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001, 1990b.
3. Wilks, Yorick. *One small head: models and theories.* In "The foundations of Artificial Intelligence: a sourcebook", Partridge, Derek and Yorick Wilks (Eds.), pp. 121-134. Cambridge, United Kingdom: Cambridge University Press, 1990.
4. Wilks, Yorick. *Decidability and Natural Language.* Mind, N.S. 80, 497-516, 1971.
5. Wilks, Yorick. *Grammar, meaning and the machine analysis of language.* London: Routledge and Keegan Paul, 1972a.
6. Mc Kevitt, Paul. *The OSCON operating system consultant.* In "Intelligent Help Systems for UNIX – Case Studies in Artificial Intelligence", Springer-Verlag Symbolic Computation Series, Peter Norvig, Wolfgang Wahlster and Robert Wilensky (Eds.), Berlin, Heidelberg: Springer-Verlag, 1990a. (Forthcoming)
7. The Byte Staff. *Product Focus: Making a case for CASE.* In Byte, December 1989, Vol. 14, No. 13, 154-179, 1989.
8. Partridge, Derek and Yorick Wilks. *Does AI have a methodology different from software engineering?.* In "The foundations of Artificial Intelligence: a sourcebook", Partridge, Derek and Yorick Wilks (Eds.), pp. 363-372. Cambridge, United Kingdom: Cambridge University Press. Also as, *Does AI have a methodology which is different from software engineering?* in Artificial Intelligence Review, 1, 111-120, 1990b.
9. Partridge, Derek. *What the computer scientist should know about AI — and vice versa.* In "Artificial Intelligence and Cognitive Science '90", (this volume) Springer-Verlag British Computer Society Workshop, Mc Tear, Michael and Creaney, Norman (Eds.), Berlin, Heidelberg: Springer-Verlag, 1990.
10. Dijkstra, E.W.. *The humble programmer.* Communications of the ACM, 15, 10, 859-866, 1972.
11. Gries, D.. *The science of programming.* Springer-Verlag, NY, 1981.
12. Hoare, C.A.R.. *The emperor's old clothes.* Communications of the ACM, 24, 2, 75-83, 1981.
13. Mc Kevitt, Paul and William C. Ogden. *Wizard-of-Oz dialogues in the computer operating systems domain.* Memoranda in Computer and Cognitive Science, MCCS-89-167, Computing Research Laboratory, Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001, 1989.
14. Sharkey, Noel E. and G.D.A. Brown. *Why AI needs an empirical foundation.* In "AI: Principles and applications", M. Yazdani (Ed.), 267-293. London, UK: Chapman-Hall, 1986.
15. Yourdon, Edward. *Techniques of program structure and design.* Engelwood Cliffs, New Jersey: Prentice-Hall, Inc., 1975.
16. Pressman, Roger S. *Software engineering: a practitioner's approach.* McGraw-Hill: New York (Second Edition), 1987.
17. Bundy, Alan and Stellan Ohlsson. *The nature of AI principles: a debate in the AISB Quarterly.* In "The foundations of Artificial Intelligence: a sourcebook", Partridge, Derek and Yorick Wilks (Eds.), pp. 135-154. Cambridge, United Kingdom: Cambridge University Press, 1990.

18. Simon, Thomas W. *Artificial methodology meets philosophy.* In "The foundations of Artificial Intelligence: a sourcebook", Partridge, Derek and Yorick Wilks (Eds.), pp. 155-164. Cambridge, United Kingdom: Cambridge University Press, 1990.
19. Narayanan, Ajit. *Why AI cannot be wrong.* In Artificial Intelligence for Society, 43-53, K.S. Gill (Ed.). Chichester, UK: John Wiley and Sons, 1986.
20. Sutcliffe, Richard. *Representing meaning using microfeatures.* In "Connectionist approaches to natural language processing", R. Reilly and N.E. Sharkey (Eds.). Hillsdale, NJ: Earlbaum, 1990.
21. Narayanan, Ajit. *On being a machine.* Volume 2, Philosophy of Artificial Intelligence. Ellis Horwood Series in Artificial Intelligence Foundations and Concepts. Sussex, England: Ellis Horwood Limited, 1990.
22. Partridge, Derek and Yorick Wilks. *The foundations of Artificial Intelligence: a sourcebook.* Cambridge, United Kingdom: Cambridge University Press, 1990a.
23. Schank, Roger. *What is AI anyway?.* In "The foundations of Artificial Intelligence: a sourcebook", Partridge, Derek and Yorick Wilks (Eds.), pp. 1-13. Cambridge, United Kingdom: Cambridge University Press, 1990.
24. Bundy, Alan. *What kind of field is AI?.* In "The foundations of Artificial Intelligence: a sourcebook", Derek Partridge and Yorick Wilks (Eds.), p. 215-222. Cambridge, United Kingdom: Cambridge University Press, 1990.
25. Dietrich, E. *Programs in the search for intelligent machines: the mistaken foundations of AI.* In "The foundations of Artificial Intelligence: a sourcebook", Derek Partridge and Yorick Wilks (Eds.), 223-233. Cambridge, United Kingdom: Cambridge University Press, 1990.
26. Sparck Jones, Karen. *What sort of thing is an AI experiment.* In "The foundations of Artificial Intelligence: a sourcebook", Partridge, Derek and Yorick Wilks (Eds.), pp. 274-285. Cambridge, United Kingdom: Cambridge University Press, 1990.
27. Hempel, C.. *Philosophy of natural science.* Prentice Hall, 1966.
28. Popper, K. R.. *Objective knowledge.* Claredon Press, 1972.
29. Marr, David. *AI: a personal view.* In "The foundations of Artificial Intelligence: a sourcebook", Derek Partridge and Yorick Wilks (Eds.), 99-107. Cambridge, United Kingdom: Cambridge University Press, 1990.
30. Marr, David. *Vision.* Freeman, 1982.
31. Connell, John L. and Linda Brice Shaffer *Structured rapid prototyping: an evolutionary approach to software development.* Engelwood Cliffs, New Jersey:Yourdon-Press Computing Series, 1989.
32. Hayes-Roth, F., D.A. Waterman and D.B. Lenat *Building expert systems.* Reading, MA: Addison-Wesley, 1983.
33. Mc Kevitt, Paul. *Formalization in an English interface to a UNIX database.* Memoranda in Computer and Cognitive Science, MCCS-86-73, Computing Research Laboratory, Dept. 3CRL, Box 30001, New Mexico State University, Las Cruces, NM 88003-0001, 1986.
34. Mc Kevitt, Paul and Yorick Wilks. *Transfer Semantics in an Operating System*

Consultant: the formalization of actions involving object transfer. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87), Vol. 1, 569-575, Milan, Italy, August, 1987.
35. Mc Kevitt, Paul and Zhaoxin Pan. *A general effect representation for Operating System Commands.* In Proceedings of the Second Irish National Conference on Artificial Intelligence and Cognitive Science (AI/CS-89), School of Computer Applications, Dublin City University (DCU), Dublin, Ireland, European Community (EC), September. Also, in "Artificial Intelligence and Cognitive Science '89", Springer-Verlag British Computer Society Workshop Series, Smeaton, Alan and Gabriel McDermott (Eds.), 68-85, Berlin, Heidelberg: Springer-Verlag, 1989.