BRUCE W. HAMILL

SYMPOSIUM ON THE ROLE OF LANGUAGE IN PROBLEM SOLVING

The JHU/APL Symposium on the Role of Language in Problem Solving convened at APL's Kossiakoff Center on October 29, 1984, for 2½ days of invited talks, contributed paper presentations, and panel discussions on the influence that problem representation and language conventions have on problem-solving effectiveness and efficiency. In attendance were more than 90 people, including representatives from Canada, Denmark, Japan, and The Netherlands.

Robert P. Rich of APL set the tone of the symposium with his comments on "The Influence of Language on Its User," and APL's George C. Weiffenbach described "Space-Age Demands for Powerful Computer Languages." In addition to their remarks, which follow this article, the program featured invited addresses by four noted academic computer scientists.

In his keynote address, Saul Amarel of Rutgers University reviewed some theoretical "Problems of Representation in Heuristic Problem Solving" that have been the focus of his artificial intelligence (AI) research efforts for more than 20 years. He identified as key issues the choice of a representation or formulation of a specific problem that avoids redundancies and irrelevancies; the choice of a representation for the situation space appropriate to the problem, including consideration of the presence of symmetries, easily traversable areas, and critical paths through this space; and the formation of "macromoves," or complex collections of steps that can be used together in solving problems of a particular kind. He emphasized the need to look at ways to solve a taxonomy of problems, rather than considering problem solving as a general task. He also suggested a link between the acquisition of expertise and the ability to reformulate problems, and he indicated that there may be a strong connection between problem reformulation and the process of theory formation.

Continuing in a theoretical vein but moving toward practical considerations, Jaime Carbonell of Carnegie-Mellon University, in his address on "AI Languages for Problem Solving," first assessed several current AI languages in terms of language criteria and desiderata for problem-solving use. He then discussed how researchers want AI languages to evolve, focusing on problem-solving reasoning strategies and knowledge-acquisition processes as the basis for determining appropriate criteria for such languages. Finally, he considered how to get from the current situation to the desired future situation. Among the reasoning strategies he described for those purposes were means-ends analysis, transformational analogy, derivational analogy, learning from examples, and reasoning from first principles. Carbonell emphasized that AI languages should permit the inclusion of multiple declarative and procedural representation formalisms, problem space transformations, different types of reasoning strategies for operating on the various representations of knowledge, and facilities for introspection, interpretation, and reflection.

Ben Shneiderman of the University of Maryland addressed more practical issues involved in "Overcoming Limitations Imposed by Current Programming Languages," with a view toward providing suggestions and guidance for research in the design of programming systems and environments. He advocated psychological experiments to determine what factors are important in the programming process. He identified among the strengths of current programming languages their precision, power, effectiveness of communication, modular decomposability, and extensibility. However, he found them to be error-prone and tedious, to require considerable training, to be lacking in structure at different levels, to have weak methods for checking errors, and to produce programs that are hard to maintain.

Shneiderman recommended several kinds of improvements, including those related to programming style (for example, commenting, structural indentation, and modular design), novel programming constructs (such as enhanced data types, high-level control structures, and integrity constraints), and the use of programming in new application areas (such as graphics programming, sound programming, and control of devices in three-dimensional space). He also advocated development of enhanced programming environments and tools, including syntax-directed editors, facilities to aid programmers in performing error-free and complete operations in the process of programming (such as automatically matching syntactic symbols like parentheses, providing templates for if-then-else statements, and carrying out the full intention of a specified command), reusable libraries of codes, program analysis tools, verification testing and debugging tools, and tools that can automatically maintain program operations when transformations are made on the structure of variables. In the future, he would like to see systems that permit end users, rather than programmers, to create programs through the use of programming tools like spreadsheets and editors; to

Dr. Hamill is a psychologist in the Mathematics and Information Science Group of the Milton S. Eisenhower Research Center.

have improved training and motivation for people who work on programs; and to move toward direct manipulation of objects, such as cursors and icons, as a means of interacting with a computer display. Finally, he distinguished between syntactic and semantic aspects of programming and discussed implications of this distinction for teaching programming and for designing programming systems.

The banquet speaker, John Carr of the University of Pennsylvania, discussed "The Future of Programming Languages" in a very entertaining, yet informative, manner. After reviewing a number of the historical developments in programming languages, he focused on the idea of a computer program as a "little person inside the box" with whom the user interacts, a concept that changes the fundamental nature of what a programming language is or should be. He observed that the Apple Macintosh graphics interface has changed the nature of interaction with computers, both because of its relatively low cost and because of the mouse-oriented user interface that replaces many keyboard operations. He also noted the availability of touch-screen interfaces, the trend toward menu-based interfaces, and the possibility of interposing a powerful search chip between a user and a complex database system, each of which has a fundamental impact on programming requirements and, thus, on the underlying nature of programming languages. The future also holds new developments in programming languages that derive from requirements of very-large-scale and ultra-large-scale integrated circuit architectures and parallel-processing control mechanisms. In addition, consideration must be given to the question of the logical equivalence of hardware and software, and the relationships among these, firmware, and "humanware" (processes of the human brain that one would like to capture in a computer). If we say that any subroutine can be put onto a chip, how does this affect programming languages? Using the idea of a foundry that can operate under the control of computer programs to produce integrated circuits representing computer programs, Carr argued that the distinction between software and hardware (i.e., chips) is becoming less clear and that this has implications for the definition of what a programming language is. Finally, he described a computer program on a chip that is generated by another program called Make Finite State Machine; if that program makes finite-state machines, then its output must be a finite-state machine in some abstract form, and one can ask whether the latter is a program and whether the program that generated it is, in fact, a programming language. The investigation of such "programming tactics" for producing programs that are chips can lead to a better understanding of constraints that exist on programming.

PHILOSOPHICAL FOUNDATIONS: THE ROLE OF REPRESENTATION IN PROBLEM SOLVING

Papers contributed to the symposium were presented in three topical sessions. In the first session, "Philosophic Foundations: The Role of Representation in Problem Solving," Amarel's invited address was followed by three papers. Stuart Hirshfield of Hamilton College presented one entitled "Programs as Symbolic Representations of Solutions to Problems," in which he discussed computer programming skills in terms of the relationship between problem-solving and linguistic abilities. He speculated that the skills needed for writing programs are equally linguistic and logical in nature, and he recommended a new approach to teaching programming to students that would emphasize instruction in both linguistic and problem-solving skills in parallel, rather than the more traditional sequence of instruction in syntax followed by attempts to write programs to solve increasingly complex problems.

John Carlton-Foss of Human-Technical Systems, Inc., discussed "Physics, Cognitive Psychology, and Computer Languages: Toward an Experimental Epistemology Using Languages as a Research Tool in the Physical Sciences." He argued that the quality of a match between computer-based products and requirements of users of those products involves a series of transformations from user "reality," to programmer reality, to system reality, to computer reality. Cognitive style influences the choice of "images" for representing problems and solutions, and it thus extends its influence to the design of specialized languages for use in solving problems in various cognitive styles.

In his paper, "AI Languages Should Be Natural," Roy Rada of the National Library of Medicine focused on ways of capturing in specialized AI languages the property of "gradualness" that occurs in the refinement of human knowledge. Gradualness appears as small changes in the specification of a problem that produce small changes in problem solution. The corresponding naturalness property of a language appears as small changes in the structural presentation of a string of symbols in the language that produce small changes in the function or pragmatics of that string. He presented examples of such changes in an expert system knowledge base that was treated as a "weighted graph," with paths through the nodes of the graph being specified and changed by means of confidence values, rules, and an edge refinement scheme.

HOW LANGUAGE CAN AFFECT ACTIONS AND SOLUTIONS

The second session was devoted to the issue of "How Language Can Affect Actions and Solutions." After Carbonell's invited address, Jack Boudreaux of the National Bureau of Standards considered "Problem Solving and the Evolution of Programming Languages." He reviewed the genealogy of programming languages and suggested that, rather than simply replacing the underlying model of computation, each succeeding generation of programming languages transfers new and more difficult cognitive functions from the programmer to the computer. This leads to the prediction that the next generation will come about not by advances in computer technology but by suc-

cessful automation of higher order cognitive functions that currently require human agents. This could lead to improved matching of computing languages with individual cognitive styles and, thus, to workstations that meet individual user's needs. He cautioned that if we should reach the point at which a computer system interacts effectively with a user through a natural language interface, it is possible that the human user will impute to the system more knowledge and reasoning ability than it actually possesses, a situation that could be dangerous in certain situations.

Noah Prywes of the University of Pennsylvania (representing his coauthors J. Baron, B. Szymanski, and E. Lock) presented "An Argument for Non-Procedural Languages" in which he suggested that nonprocedural languages are closer to the "natural" way of representing most problems than are procedural (sequential) languages, especially inasmuch as they save the user the trouble of working out the order in which assignments to variables will be made, since multiple assignments cannot be made to a single variable. Features of nonprocedural languages include unambiguous semantics, explicitness, ease of debugging, support in problem decomposition, and a general approach to problem solving in which the system translates a problem statement into results rather than following a specified sequential set of procedures. Experiments in which students solved problems in the authors' nonprocedural MODEL system indicate that the time required to program certain complex systems, such as an accounting system, is considerably less than that required in standard procedural languages, although the number of programming errors does not differ.

The paper by Thomas Strothotte of the University of Waterloo concerned "The Use of the Subjunctive in Problem Solving." Observing that current programming languages use only the present tense, while natural languages use a rich variety of tenses and moods for personal communication and problem solving, he suggested adding the subjunctive mood to programming languages as a feature to facilitate problem solving. The subjunctive would permit one to construct such statements as, "If I take another step, will I fall off the cliff?" Current languages only permit solutions of this problem in the form, "Take a step. Did I fall off the cliff? If yes, go back a step; if no, take another step." He has implemented an IF WOULDBE (expression) THEN. . .ELSE. . .primitive construct in PASCAL to capture the look-ahead feature of the subjunctive, and he argued that it removes some of the awkwardness of using algorithmic languages.

OVERCOMING LIMITATIONS IMPOSED BY CURRENT PROGRAMMING LANGUAGES

The third session, which concluded with Shneiderman's invited address, was devoted to means of "Overcoming Limitations Imposed by Current Programming Languages." Anand Desai of Digital Equipment Corporation (representing his coauthor Robert Jernigan of Decision Resource Systems) described "XIMM—An Expert System for Idle Materials Management: Logic Programming for Corporate Strategies." The expert rule base of XIMM, a system that plans redistribution, upgrading, and dismantling for parts of various Digital Equipment Corporation computer products, is written in APLLOG, an integration of PROLOG and APL languages. APLLOG supports "scripts" (sets of conditions and rules to make sorts through databases and to produce and output results), rules (in the form of Horn clauses), a knowledge base, functional predicates, macro commands, projection functions, ordinary functions, and an editor. He demonstrated the effects of these features in a detailed discussion of XIMM output.

Aaron Temin of the University of Texas at Austin (representing his coauthor Elaine Rich also of the University of Texas at Austin) discussed "MIRROR: A Language for Representing Programs for Reasoning." MIRROR is a programming language (or a "program representation language") whose goal is to make explicit in programs those things that an on-line help system would need to know. The help system in question is for the text editor SCRIBE; both the help system and SCRIBE are written in MIRROR, which itself is written in LISP. Although the help system is very limited at present, it is being developed in such a way as to enable it eventually to explain cause-and-effect relations, comparisons, look-ups, and command syntax through user modeling and the use of several program design techniques that will facilitate operation of an intelligent help system.

In his paper entitled "APL—A Pictorial Language," Ross Bettinger of The Mitre Corporation advocated the use of graphic concepts in problem solving. Noting a distinction in current research on human brain function between verbal-analytic-linear and nonverbal-gestalt-nonlinear modes of thinking that are characteristic of the left and right hemispheres, respectively, of the brain, he suggested that most programming languages depend heavily on "left-brain thinking" and do not use right hemisphere capabilities, while APL (and to some extent LISP) incorporates both left- and right-brain modalities of thought, making it a superior language in which to formulate problems for computer solution. He demonstrated through examples the use of geometric (pictorial), rather than algebraic, thinking in solving problems using the APL language.

James Ryan of Analogic Corporation (representing his coauthor Michael Berry) described "Threaded Workspaces—An Environment that Facilitates the Programming and Execution of Large and Complex Application Systems." Threaded workspaces are a means of organizing large, complex systems into well-defined, easily understood, and easily connected program modules through the use of named contexts and ways of linking and binding variables from different contexts among modules. This programming environment permits the construction and use of a library of program modules that can be combined in an ap-

plication system of cooperating, possibly concurrent, processes.

Naomichi Sueda of Toshiba Corporation (representing his coauthors Shinichi Honiden, Yoichi Kusui, and Kuzuo Mikame) discussed "PROLOG Application in Software Components Reuse." The concept of software components reuse has been incorporated in an image-processing logic simulator designed to improve efficiency in checking algorithms for image processing. The simulator operates on a system of some 350 software components in the Subroutine Package for Image Data Enhancement and Recognition (SPIDER). The logic simulator chooses components of SPIDER that can be used without reprogramming and sets up combinations of those components automatically. Parameter attributes of each component are treated as knowledge, and PROLOG is used to infer relationships among parameters through the application of rules defined for that purpose.

A paper entitled "Solving Graph Problems Using LOGRAPH" was presented by Tomasz Pietrzykowski of the Technical University of Nova Scotia (representing his coauthor P. T. Cox). LOGRAPH, a pictorial language implemented in PROLOG, is designed to overcome difficulties in representing complex problems and algorithms that arise in textual descriptions. It permits a graphical form of a problem to be represented, not merely as a heuristic aid to the user, but as pictures on a screen that can be directly executed. The pictures, which resemble flow charts, consist of frames with compartments (variables) that are connected by lines that represent the operational relations among them.

Bruce Blum of APL discussed "Language, Problem Solving, and System Development." Focusing on requirements for the development of new software tools, he emphasized the importance of finding errors early, noting that most errors occur in the system design phase. He stated that programmer productivity is a function of the size of the program being coded. He also pointed out that maintenance is the major part of software cost, with coding accounting for only about 20 percent of the time that will eventually be spent on a software system. His approach to system development is to work at a high enough level that he can be concerned with issues in the problem domain rather than with coding issues. His TEDIUM system, which is written in MUMPS, is designed to enable the user to work at that level so that effective application systems can be developed by domain experts with relatively little experience or interest in programming languages.

PANEL SESSIONS

The symposium culminated in two panel sessions. The first was devoted to the discussion of "Language Requirements for Effective and Efficient Problem Solving." This panel was chaired by David Barstow of Schlumberger-Doll Research; panelists were Jaime Carbonell, John Carlton-Foss, Adin Falkoff of IBM/T. J. Watson Research Center, and Andrew Goldfinger of APL. Barstow focused attention on three issues:

1. The effectiveness or efficiency with which the user can communicate the problem, or how easy it is to state a problem;
2. How languages can help in the solution of a problem, that is, what is the nature of the solution process and what features of languages might help or hinder that process;
3. If there is some measure of the quality of the result of the solution process, what language features might help us find the best or worst of alternative solutions.

Discussion by panelists and others ranged across many topics, including:

1. The effects of differences among the world's natural languages on the ways their speakers think;
2. The relationship between individual differences in personality and the choice of programming languages;
3. The interaction among the person trying to solve a problem, the problem itself, and the domain or context in which the problem occurs;
4. Suggested extensions of current programming language capabilities to support specialized problem-solving applications;
5. Aspects of the APL language that seem to produce changes in the way people think about problems—arrays, functional notation, operators for function transformation, and shared variables;
6. The importance of programming environments to system design and development, especially for research systems;
7. How to specify and bring in implicit knowledge for use in problem solving by a system—the "frame problem" in AI;
8. Possible efficiency advantages of "wide-spectrum" languages in which components that are composed may be either primitive or abstract;
9. Only primitives in programming languages, with other kinds of functions being developed from the primitive for various applications in order to avoid erroneous assumptions about the purposes and operations of complex "black-box" functions;
10. Building high-level languages by combining primitive-based languages in hierarchical structures;
11. The importance of documentation for programming languages;
12. The influence of PROLOG on how one thinks about solving problems;
13. The prospect of new modalities of thought deriving from programming languages;
14. Difficulties involved in understanding and teaching the concept of recursion;
15. Cooperative use of the abilities of the human

and the computer system so that each performs those functions that it does best;
16. Inheritance hierarchies as tools for organizing problems for solution.

The second panel session considered issues related to "Comparative Application of Computer Languages to Practical Problems." The panel was chaired by Bruce Blum; panelists were Roy Rada, Elaine Rich, Jean Sammet of IBM, and Ben Shneiderman. The range of topics discussed by the panelists and others included

1. Criteria and measures (or the lack thereof) for comparing programming languages;
2. Software engineering;
3. Programming as a productive activity;
4. Transportability of software;
5. The problem of investment in obsolescent languages;
6. The unpredictability of software needs for future hardware;
7. What a programming language is;
8. Languages for specialized application areas, i.e., special-purpose languages;
9. Functional capability and tailored style as fundamental needs in special-purpose languages for solving problems in particular application areas;
10. Identifying relevant literature for guidance in how to make the relationship between computers and their users more natural;
11. Software reusability;
12. High-level programming requirements;
13. Facilities for direct manipulation of objects on a computer terminal;
14. Technological limitations on how problems are represented for computer solution;
15. The difference between a language and its implementation;
16. Use of natural languages (e.g., English) for programming;
17. Use of "natural formal notations" (as opposed to natural languages) for programming that are natural to the problem area or domain.

SUMMARY

Finally, Jack Boudreaux offered a summary of the symposium in which he identified some of the principal themes that he would like to see addressed further at future symposia and conferences. The first such theme was that of defining a paradigm within which to consider the role and function of computers and computer-based systems. Three possible alternatives are the computer as an assistant, the computer as a user environment, and the computer as a tool kit. The second theme concerned methods for eliciting and representing what users know and believe about "objects" that are important in field-specific user domains. His third theme was the notion of program schemata and plans that mediate between what the programmer knows about a programming language and what he knows about the application domain. The fourth theme he identified was the distinction between language as text and language as graphic sign, or the distinction between linear and two-dimensional means of communication.

The Symposium on the Role of Language in Problem Solving was characterized by very open discussions of the contributed papers and invited addresses, lively exchanges among panelists and audience participants, and a high degree of animated interaction among all participants. The general consensus was that this was a very successful symposium and that another should be planned for the future to pursue this topic.

The symposium proceedings will be published by the North-Holland Publishing Company as *The Role of Language in Problem Solving—I* in 1985.

ACKNOWLEDGMENTS—I thank my cochairman David Weintraub, program chairman Robert Jernigan, support coordinator Barbara Northrop, publicity coordinator Constance Finney, and members of the program committee and the support staff for their excellent performance in planning and executing the symposium. Special thanks go to George Weiffenbach and Robert Rich for their early interest and continued support and to William Guier and Daniel Brocklebank for their advice and counsel during planning. Finally, I thank the Applied Physics Laboratory for its financial and facilities support, without which the symposium could not have been held.

THE INFLUENCE OF LANGUAGE ON ITS USER[1]

ROBERT P. RICH

It is still true (and I suspect that it will remain so for some time) that the most effective solvers of problems still are people, and the thrust (as I take it) of the meeting is to show that choice of a suitable notation has an effect on the way we approach and solve problems. In case there are any who do not believe that, I have a very simple example, as shown in the figure. Until about the 12th century, people used Roman numerals for commercial bookkeeping and any other purposes that might occur to them, and I believe the books of the Medici were kept in Roman numerals as a matter of law until as late as 1499. So it took 500 years for Europe to move from the notational system shown in the top panel down to that shown in the lower left-hand corner.

In the top panel I show the multiplication of 47 by 53 using the duplation method—not in the form in which the Romans, via the Greeks, originally got it

Dr. Rich is Supervisor of the Data Processing Branch at APL.

XLVII	LIII
XXIII	CVI
XI	CCXII
V	CDXXIV
II	~~DCCCXLVIII~~
I	MDCXCVI
	MMCDXCI

```
  53
X 47
 371
212
2491
```

$$(X - Y)(X + Y) = X^2 - Y^2$$

$$(50 - 3)(50 + 3) = 2500 - 9$$

$$= 2491$$

from the Egyptians, but in a form that may be more familiar to the reader. The top panel shows two columns of Roman numerals. The left gives the results of successively halving the number 47, throwing away the remainders. The right column shows, above the line, the results of successively doubling 53. Each number in the right column is crossed out if the corresponding number in the left column is even; in this example only one such crossed-out number occurs. Below the line in the right column is the sum of the numbers above the line that did not get crossed out, namely, the product 2491 of the numbers 47 and 53 we started out with.

All of the operations involved—division by two, doubling, addition, and recognition of odd numbers—are easily performed on the abacus. But the whole process is a cumbersome one, especially when compared with the more familiar decimal form of the same computation, as shown in the lower left corner. Here we show that for certain simple calculations, a choice of notation permits you to do away with the Roman abacus (which was a pretty good-sized grooved rock; the ones that have been recovered weigh anywhere from 50 to 200 pounds). The paper and pencil, all that is required for the Arabic numerals, were great advances in portability, and portability, of course, as applied to personal computers, is still very much in the news. But we are still performing the same calculation in both of those panels.

As we move over to the panel at the lower right, we are, in principle, doing the same thing, but we are taking a very different approach to our problem; namely, we recognize that this is a useful special case of the general situation that the first two panels address. We have an algebraic formula that happens to fit the two numbers that I have picked and we get the same answer, 2491, in our heads. Now, I think that during the next several days we are going to be shown analogs of both the decimal and the algebraic approaches and three or four still higher levels of ways in which language can help us solve our problems.

I would like to start out by saying that natural language and human culture are obviously very closely related, and human culture is obviously closely related to thinking. So we have an influence from language to culture to thinking, and this influence (or at least a specific form of it) is known as the "Whorf-Sapir Hypothesis." In Edward Sapir's words (as quoted by his student, the insurance adjuster Benjamin Lee Whorf):

> Human beings do not live in the objective world alone, nor alone in the world of social activity as ordinarily understood, but are very much at the mercy of the particular language which has become the medium of expression for their society. . . . We see and hear and otherwise experience very largely as we do because the language habits of our community predispose certain choices of interpretation.

In the words of Benjamin Lee Whorf, who also has his name attached to this hypothesis that we think the way we do because we speak the way we do:

> It was found [as a result of his study of the Hopi language] that the background linguistic system (in other words, the grammar) of each language is not merely a reproducing instrument for voicing ideas, but rather is itself the shaper of ideas, the program and guide for the individual's mental activity, for his analysis of impressions, for his synthesis of his mental stock in trade. Formulation of ideas is not an independent process, strictly rational in the old sense, but is part of a particular grammar, and differs, from slightly to greatly, between different grammars.

The Whorf hypothesis has been widely discussed in the linguistics community, discussion that is motivated partly by the fact that the theory is very difficult to demonstrate and partly by the fact that because Whorf was an insurance adjuster without formal training in linguistics, anything he said had to be seriously attacked as a matter of professional pride. Incidentally, I might point out that we have some of that in the programming language community, but none of you have ever noticed it, I am sure.

There is a real question about the way people think and the influence of language on the process. Jacques Hadamard, in his delightful little book on *The Psychology of Invention in the Mathematical Field*, devotes a major chapter (the sixth) to the question of whether people in fact do use language in their thinking or do not. I will not try to reproduce the chapter, but I do recommend the book and the chapter in particular to those who are interested in the subject. In the chapter, he gives a champion for each opinion; much of my further discussion involves the men and their approaches as Hadamard presents them in his book.

Max Müller, the etymologist, felt that there was no possible way for anybody to think except by using words, while Francis Galton, the biostatistician (a rather unfortunate choice, I think), was the one who pointed out that a lot of people do think without words. I think that this is nicely resolved by Edward Wilson in the Autumn 1984 issue of *The American Scholar*:

There is a real question about the way people think and the influence of language on the process.

> The symbols of art, music, and language freight power well beyond their outward and literal meanings. So each one also condenses large quantities of information. Just as mathematical equations allow us to move swiftly across large amounts of knowledge and spring into the unknown, the symbols of art gather human experience into novel forms in order to evoke a more intense perception in others. Human beings live—literally live, if life is equated with the mind—by symbols, particularly words, because the brain is constructed to process information almost exclusively in their terms.

He is taking the Max Müller approach to the question, of course, and the other approaches you can find well spelled out in Hadamard's chapter. The whole question, I think, is put into perspective by Roman Jacobsen (as quoted by Hadamard), just as he has put so many other questions into proper perspective:

> Signs are a necessary support of thought. For socialized thought (stage of communication), and for the thought which is being socialized (stage of formulation), the most usual system of signs is language properly called; but internal thought, especially when creative, willingly uses other systems of signs which are more flexible, less standardized than language and leave more liberty, more dynamism to creative thought.

William Rowan Hamilton put it in a nice analogy that Hadamard describes in his book: Think of a person digging a tunnel in a sandbank. He finds that after he has dug about a foot, the ceiling starts caving in; he has to get out of the way and let a mason build arches or somebody do some other form of shoring so that he can come back into the tunnel and dig the next foot. So we have two different people doing different things: the fellow with the shovel digging the sand out of the way and the fellow with the trowel and mortar building up the arches behind him. If we think of the dynamism of creative thought as the fellow digging away at the problem and then of the transformation and communication of that thought as the mason building the arches, we realize that there can be a single activity carried out by very different means and that the final result (namely, a tunnel you can safely walk through in one case or a paper in a journal in another) really requires both kinds of work. What seemed to be an unsolvable problem turns out to be no problem at all. I refer once again to the sequence of panels in my figure; this is what we really mean, I think, by problem solving: To the extent that we can find that the problem we have to solve is, in fact, no problem or a trivial one, we have taken advantage of our intelligence; perhaps we can teach our machines to take advantage of their intelligence in the same way.

I am reminded of one of H. H. Munro's little essays (writing as Saki). I will not attempt to quote it because it is a bit too long to read every word, but again, I refer you to the full text. He had written a book that had gotten some slight renown, and a friend of the family asked where she could get a copy. He pointed out that having recourse to an ironmonger or a greengrocer would entail delay and disappointment and suggested that she visit a bookshop. She met him at a private view a couple of weeks later and said "It is all right, I borrowed it from your aunt." This is another example of essentially working around a problem rather than actually addressing it, and I recommend to all of the intelligence people here, artificial and natural, that you look for ways of avoiding your problem before you spend a lot of time solving it.

Let me summarize this phase of my talk, namely, the influence of natural language on culture and therefore on people, by a little bit of doggerel.

> The Irish had no word for "no,"
> The Romans none for "yes,"
> Which language best helped empire grow
> Is easy to assess.

I now move to the second part of my talk, which deals not with natural language and natural people but with programming languages and programmers. (That did not come out exactly the way I meant, but I guess it made the point clearly.) I would like to introduce the topic by taking a far-out position from which we can perhaps swing back toward the center. This is one taken by David Bolter in *Turing's Man*, another recent book that I recommend to your attention and that requires careful and critical reading. "The whole course of linguistic philosophy from Leibnitz to the positivists seems to culminate in the computer, where symbols are drained of connotations and given meanings solely by initial definition and by syntactic relations to other symbols." One of the reasons why I recommend that you read Bolter critically is exemplified here. Those of us who have become fluent in a programming language realize that, although what Bolter says is strictly true when we first open the manual, it rapidly becomes untrue as we become fluent in the language and begin thinking in it (as Whorf would say) because the symbols in artificial languages pick up their own connotations as well as their denotations just as symbols do in natural languages.

I think that a good way of putting this is Richard Conner's statement, "A properly trained programmer

thinks primarily in terms of programming, only secondarily in terms of a particular language." That really is going a bit toward Galton's view of the Whorf hypothesis. We can move back to the other side by means of a sentence from Dijkstra's *Discipline of Programming,* "A most important, but also a most elusive, aspect of any tool is its influence on the habits of those who trained themselves in its use. If the tool is a programming language, this influence is—whether we like it or not—an influence on our thinking habits." This could be a straight quotation from Whorf and subject to the same arguments pro and con on both sides. I am not going to say very much about specific languages because there are a number of people here who would protest violently against any specific statement I might make about a specific language. I will quote Conner again, from his nice little essay, "Happy 25th Birthday, COBOL," in *Computerworld*: "Although we think of COBOL as a language, this discussion [that is, his, not mine] will treat it as something more—a mentality, if you will." Benjamin Lee Whorf again.

I have an interesting example of the direct influence of culture rather than language on behavior; this is again from Dijkstra. One of the problems he set was to arrange a line of marbles in the order of the colors of the Dutch national flag, which is red, white, and blue. The subject is given a device that can pick up a marble, look at it and determine its color, and move it to one place or another. However, it is very expensive to do this so we do not want to do it twice to the same marble. The solution to the problem is pretty straightforward: The subject takes the groove in which the marbles are and says, "I am going to put all the red ones to the left, then the white ones, and then the blue ones." Thus, as he picks up each marble and determines its color, he has the problem of where in the groove to put it. When Dijkstra asked his students, who were either Dutch or American, which marble to inspect first, their suggestion was always "the leftmost one." He says: "I had the idea that this preference could be traced to our habit of reading from left to right. Later I encountered students that suggested first the rightmost one, one was an Israeli computing scientist, the other was of Syrian origin." So here we have a secondary influence of language, namely, the direction in which you write it does have at least this modest effect on our thinking of various kinds of problems.

Why is it easy for people to continue to disagree on the effectiveness of particular languages? According to B. A. Sheil's article on the psychology of programming in *Computer Surveys*:

> As practiced by computer science, the study of programming is an unholy mixture of mathematics, literary criticism and folklore. However, despite the stylistic variation, the claims that are made are all basically psychological; that is, that programming done in such and such a manner will be easier, faster, less prone to error or whatever. . . . Sadly, however, psychological data have been at best a minor factor in these debates.

With that very real understatement of the situation, I will terminate my comments.

BIBLIOGRAPHY

J. D. Bolter, *Turing's Man: Western Culture in the Computer Age,* University of North Carolina Press, Chapel Hill (1984).

R. L. Conner, "Happy 25th Birthday, COBOL," *Computerworld* (Apr 5, 1984).

R. L. Conner, *Computerworld* (Sep 10, 1984).

E. W. Dijkstra, *A Discipline of Programming,* Prentice-Hall, Englewood Cliffs, N.J. (1976).

J. Hadamard, *The Psychology of Invention in the Mathematical Field,* Princeton University Press (1945).

M. Müller, *Three Introductory Lectures on the Science of Thought,* London (1887).

H. H. Munro, "The Sex That Doesn't Shop," *The Complete Works of Saki,* Doubleday & Co., Garden City (1976).

B. A. Sheil, "The Psychological Study of Programming," *Computing Surveys* **13**, 101-120 (1981).

B. L. Whorf, *Language, Thought, and Reality,* J. B. Carroll, ed., MIT Press (1956).

E. Wilson, "The Drive to Discovery," *American Scholar,* Autumn (1984).

SPACE-AGE DEMANDS FOR POWERFUL COMPUTER LANGUAGES[1]

GEORGE C. WEIFFENBACH

The title of my comments is somewhat more grandiose than is appropriate and my real intention is somewhat more modest. In a broad sense, I represent the user community, and more particularly, I view myself as a broker for users, whether they are interested in scientific applications or other fields in the space arena. I'll restrict most of my remarks to things I am familiar with, activities we are involved in now, and then I will take a flier and do a little speculating at the end.

I would like to address four general areas; they look somewhat different, but there are relationships. I will address first implantable medical devices, then scien-

Dr. Weiffenbach was formerly head of the APL Space Department and is now a Senior Fellow at APL.

With satellites, more emphasis has got to be placed on correct design than in almost any other case....there is no feedback through large production.

tific computing, spacecraft satellite design and manufacturing, and last, smart spacecraft.

Under the technology utilization program of the National Aeronautics and Space Administration, APL has been involved in a number of projects to develop devices that are implanted in human beings. The work has been done in conjunction with the Johns Hopkins Medical School, the Massachusetts General Hospital, the Mayo Clinic, and a number of other organizations. Some years ago, we started with pacemakers, which essentially had no smarts, and more recently we have worked on a number of other things. As a sampling, there are defibrillators with very primitive intelligence; that is, they can observe the heart rate and note when it goes into fibrillation—when it is arrested in effect—and then shock the heart, all internally. Currently it looks like hypertension is going to be the first closed-loop system that we will be able to establish, a closed loop in the sense that it will measure a person's blood pressure and, in response to that, will inject medicine into the bloodstream. In addition, there are an artificial sphincter, an insulin pump (we actually have insulin pumps in several dogs in clinical trials with an implantable device that has a fairly competent computer), and pumps to inject morphine into the spinal column. The last may very well be our first human implant, because it is hard to test those on animals. Without going into a lot of detail, all these cases present opportunities for some very smart implantable devices. I don't think it takes too much imagination to think of the things one would like to be able to do that would involve expert systems programmed in very-large-scale integrated circuits for implantable devices.

In scientific computing (or information processing if you will) we are involved in three areas of research: space physics (magnetohydrodynamics or plasma physics), physical oceanography, and solid-earth geophysics. These all deal with very complex and heterogeneous systems and already have enormous databases that are being augmented at an accelerating rate. Examples are the imaging devices put in orbit for both physical oceanography and solid-earth geophysics, which produce data at prodigious rates. Another interesting characteristic all three areas share is that they are at the point where current theory is not adequate, and not adequate in very important ways, which is shown by the fact that we have already seen a number of phenomena that are predicted only through the introduction of nonlinear theory.

Furthermore, we have a data glut that you would not believe. As one example, for the imaging radar that was put into orbit on the SEASAT satellite in 1979, only 40 percent of the data has been processed and no more than 10 percent has been analyzed; that satellite lived for only 90 days. What we do not have is an *information* glut; I think the role of the techniques discussed at this symposium is obvious.

In the third area I wish to address, namely, spacecraft design and fabrication, we have all the normal activities that come with running a business: accounting, management information systems, and the like. It is already quite clear that, even in this rather normal kind of enterprise, we come up short in the computing systems that we have available to address these issues. But in the satellite business, we have other routine activities, e.g., inventory systems, that are not quite standard. Because of the enormous cost of designing, building, and launching satellites, and because we really want to achieve the most reliable possible system, we keep detailed birth-to-death records on each of the many thousands of components on the satellite. Obviously, record keeping is a serious problem for us. Robert Jernigan and some other people here have put together an inventory system; they do not have it quite altogether—mainly a funding limitation—but it has pulled us out of a real choke point. We were so backed up that we were losing schedule, and that is a very costly situation.

Looking at the space hardware design process, we have a shortfall in computer-aided design, computer-aided manufacturing, and computer-aided engineering. With satellites, more emphasis has got to be placed on correct design than in almost any other case. The reason is that there really is no mass production. We cannot rely on a million customers out there somewhere to debug what we make—there is no feedback through large production. There is no graceful way to see what we have produced, note how it functions, and then correct it. Every time we miss the boat there are enormous cost and schedule problems. We are not nearly where we would like to be in space hardware design tools. That is the existing situation; I hope that I am getting across the message that we urgently need the tools that you people can provide.

Looking into the future toward smart spacecraft, I get even more interested in pushing your art. We have already seen the past impact of computers throughout our society. In space technology, we are going to see a total turnaround. Satellites have been kind of a "gee-whiz" business; there is a lot of glamour attached to it. We see the enormous rockets go off and it is very impressive. A less spectacular but more impressive future is in store. In less than ten years, I am convinced that we will have the ability to design an information processor, a computer, that can be put into a satellite that will easily have all the power of a CRAY and will

almost certainly be even smarter. One of the driving factors that I have noted—and you do not have to be very sharp to see this happening—is the enormous progress that is being made in the hardware side of the computing business. We already have chips on the market with half a million components on them, and there does not seem to be any fundamental limitation to increasing the number of components you can put on a silicon chip by a couple of orders of magnitude. It is fascinating that the Japanese are working on chips of this kind, but with perhaps 40 layers on them. The hardware is coming. It has an enormous impetus behind it for all kinds of reasons.

What I have not yet seen is comparable progress in our ability to exploit the hardware, which clearly means computer architectures and programming. Some very simple arithmetic, verified by a lot of experience, tells us that if we are going to design a CRAY for a satellite, the number of man-years needed to design the architecture and to turn it into a useful device is enormous. When we look further downstream as the hardware gets more potent, today's conventional approaches are simply going to become impossible. I do not mean only from the standpoint of the length of time and the number of people involved; if we do not find different, more effective ways to carry out this process, we will never get there. I think everybody here must be very much aware of the futility of adding more and more people into a programming task of any kind (and designing the architecture of a smart computer has got to be one of the most sophisticated of these tasks). Sooner or later you reach a stage of diminishing returns, where you simply never arrive at the end point.

The potential that I can see in future smart, autonomous satellites is mind-boggling. I doubt that anybody can really predict the manifold uses that future satellite-borne computers can be put to. But I am convinced that they will, in fact, allow us to do things that we are totally unable to do today in terms of satellite autonomy, reliability, and information processing. I count on this community to provide the means by which we can bring this about.

REFERENCE

[1] The comments by R. P. Rich and G. C. Weiffenbach will appear in a forthcoming Proceedings volume of the symposium on *The Role of Language in Problem Solving—I*, R. Jernigan, B. W. Hamill, and D. M. Weintraub, eds., North-Holland Publishing Co., Amsterdam (in press). Reprinted by permission.