PERSPECTIVES ON COMPUTER SCIENCE

From the 10th Anniversary Symposium at the Computer Science Department, Carnegie-Mellon University

Edited by

ANITA K. JONES

Department of Computer Science
Carnegie-Mellon University
Schenley Park
Pittsburgh, Pennsylvania

ACADEMIC PRESS New York San Francisco London 1977

A Subsidiary of Harcourt Brace Jovanovich, Publishers

ACADEMIC PRESS, INC.
111 Fifth Avenue, New York, New York 10003

United Kingdom Edition published by
ACADEMIC PRESS, INC. (LONDON) LTD.
24/28 Oval Road, London NW1

Library of Congress Cataloging in Publication Data

Main entry under title:

Perspectives on computer science.

(ACM monograph series)
Held Oct. 6-8, 1975.
Includes bibliographies.
1. Electronic data processing–Congresses.
2. Electronic digital computers–Programming–
Congress. 3. Programming languages (Electronic
computers)–Congresses. I. Jones, Anita K.
II. Carnegie-Mellon University. Computer Science
Dept. III. Series: Association for Computing
Machinery. ACM monograph series.
QA75.5.P47 001.6'4 77-8319
ISBN 0–12–389450–6

PRINTED IN THE UNITED STATES OF AMERICA

List of Contributors

Numbers in parentheses indicate the pages on which the authors' contributions begin.

C. GORDON BELL (7), Digital Equipment Corporation, Maynard, Massachusetts and Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

HANS BERLINER (39), Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

RICHARD E. FIKES* (63), Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California

A. N. HABERMANN (77), Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

RENATO ITURRIAGA (91), Coordinor General del Sistema Nacional de Informacion, Palacio Nacional, Mexico City, Mexico

ZOHAR MANNA† (103), Applied Mathematics Department, The Weizmann Institute of Science, Rehovot, Israel

ALBERT R. MEYER (125), Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts

ALLEN NEWELL (147, 183), Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania

ALAN PERLIS (1), Deparement of Computer Science, Yale University, New Haven, Connecticut

R. REDDY (183), Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

GEORGE ROBERTSON (147), Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania

ADI SHAMIR‡ (103), Applied Mathematics Department, The Weizmann Institute of Science, Rehovot, Israel

MICHAEL IAN SHAMOS (125), Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

*Present address: Xerox Corporation, Palo Alto Research Center, Palo Alto, California
†Present address: Artificial Intelligence Laboratory, Stanford University, Stanford, California.
‡ Present address: Computer Science Department, University of Warwick, Coventry, England.

HERBERT A. SIMON (199), Departments of Computer Science and Psychology, Carnegie-Mellon University, Pittsburgh, Pennsylvania
WILLIAM A. WULF (217), Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

Preface

This volume is a record of the Carnegie-Mellon University computer science department's 10th Anniversary Symposium, held October 6–8, 1975 in Pittsburgh, Pennsylvania. The symposium was a celebration of the 10 years of research and education of the computer science department. Founded in the summer of 1965, it was a natural outgrowth of an interdisciplinary systems and communication sciences program. Throughout its life the department has had a major commitment to research. So it seemed appropriate that any celebration of its 10th anniversary should be a vehicle for continuing to do science, a medium through which researchers could communicate their current concerns and ideas.

We wished to have the symposium at the site of the department, but physical facilities limited the number of people we could accommodate. So we decided to invite all those who have been members of the department "family" to participate. This included students, faculty members, research associates, relatively long-term visitors (i.e., here at CMU for a month or more), and some friends and benefactors of the department.

The technical program consisted of invited papers, panel discussions, contributed talks, and demonstrations of some ongoing research at CMU. It is mainly the invited papers that are reproduced here, although we have tried to chronicle the flavor of the whole symposium by listing the demonstrations and contributed talks and including excerpts from the panel discussions.

One of the invited talks is the "riverboat speech" delivered at the symposium banquet held aboard the riverboat, the Gateway Clipper Showboat, while it plied the waters around Pittsburgh. The speaker was Alan Perlis, the first head of the computer science department.

The symposium focused on diverse topics, and is a reflection of the interests that have remained strong at CMU during the past 10 and more years. Many of the papers provide perspectives. Gordon Bell looks back on the experiences of the design and production of the PDP-11 family of machines. Raj Reddy and Allen Newell look ahead, seeking multiplicative speedup in programs performing AI tasks; Bill Wulf considers the next generation of programming languages. The symposium also provided an opportunity to analyze our own day-to-day experience and inquire after the methods of doing computer science research here at CMU. In particular, Allen Newell's paper and the related "little engines" panel provided a forum for discussing the experiences of a number of research groups that have had different interests and goals, but that cooperate and interact as they share the use of the C.mmp hardware.

We used this volume itself as an excuse to further develop our department's facilities for document production. With only two exceptions, the papers in this book were delivered to us in digitized form. A set of existing document formatting programs was augmented and new ones developed, mainly by Brian Reid, so that we could build the command string to drive a photocomposing system to "typeset" text. That is how this volume was produced. The troubles attendant on automating a portion of the book production process has made the symposium volume disappointingly late in being published. Now, of course, we know how to produce such a book right!

In a lighter vein the symposium was an excuse to sponsor a contest to find a department logo that would graphically represent our interests and activities. The winning logo, designed by Peter Hibbard, appears on the dust jacket of this volume (as well as on scores of T-shirts, following a well-established local custom). In the *Science* article which Alan Perlis mentions in his riverboat speech, computer science is described as the study of the phenomena surrounding computers. Perhaps the logo bears this attitude out. One of the few things we all have in common is the computing engine. Our logo has a one-core memory, threaded on the read, write, and sense wires, and surrounded on four sides by the initials "CMU".

This record of the symposium is a commemorative volume as well as a scientific source. It conveys the directions and the flavor of the department's collective research interests. It reflects our basically pragmatic goal-oriented bias and our bent toward system development as a vehicle to do science. But mainly it is a product of and a testimonial to those who find "fun" in computer science.

I would like to thank the many members of the department who contributed to the success of the symposium in one way or another, particularly George Robertson and Guy Almes. My grateful thanks also to Brian Reid and to David Jefferson, who contributed greatly to the production of this book. The Academic Press production staff were ever patient and helpful, especially with the confusing foibles of computerland.

Foreword

EDUCATION AND RESEARCH AT CMU

When did computer science start at CMU?

Was it in 1956 when Herb Simon told his class in the Graduate School of Industrial Administration that over the Christmas holidays he and Allen Newell had invented a thinking machine? That was the very same year Alan Perlis arrived to start a computing center. Or was it in 1961 when Newell, Perlis, and Simon, along with Abe Lavi of EE and Bert Green and Lee Gregg of Psychology, started an interdisciplinary graduate program in Systems and Communication Sciences? Perhaps it was in 1965 with the founding of the Computer Science Department. That is certainly the founding date we used in computing our 10th anniversary. The precise date is not important, except for purposes of celebration. What does matter is that from the beginning a serious and continuing commitment was made here to excellence in computer science.

In some ways today's department is very different from the department in 1965. With several very significant exceptions the faculty is new. Yet the tradition of leadership in computer science teaching and research continues. I could cite statistics: publications, honors, research contracts, indicators of student quality, and positions held by our graduates. But these can be found in our annual Computer Science Research Review. Here I prefer to list some of the unique aspects which we have evolved and of which I am particularly proud:

(1) The Immigration Course, a six-week acculturation in computer science which gives our entering graduate students a common foundation.

(2) The total integration of research and teaching: From the time they leave the Immigration Course, students spend at least half their time on research. Of course, as the students enter "thesis mode," this becomes essentially full-time.

(3) The "Black Friday" Review: This semiannual evaluation by the entire faculty of every student in the department permits us to monitor

progress and apply appropriate forcing functions while allowing students great individual freedom.

It is almost mandatory at an anniversary to consider the future. I will confine myself to two issues, one concerning the field and the other the department.

In many ways this has been a relatively easy decade for computer science. We have been the bright new science with many more good research problems than people to solve them and many more good positions than people to fill them. That is changing rapidly as the discipline matures and takes its place as one of the major sciences. I welcome that maturity even though I realize that in many ways our lives will be harder. An important issue for the maturing discipline is to maintain the vigor and excitement which first drew us to the field.

I'll turn to what I regard as a central issue for the department. Human institutions ranging from baseball teams to empires first achieve and then lose excellence. How can the department maintain and even strengthen its excellence over a long period of time? I believe this can be done only by continuing to select faculty who are broad and able to move with and, indeed, to lead the field. We must continually look at the brightest people and bring at least some of them here; and we must do this over a long period of time. This is particularly crucial in a field which gives every sign of continuing to evolve rapidly. It seems to me that the answer is to continue to grow but slowly, always exercising the highest standards. This is almost a cliche and yet it is the key to continued excellence. I want the department in 20 years to still enjoy some of the flexibility in bringing in new people that we have today.

Finally, a word about this symposium. I have never been prouder of the department than during the days we were preparing for it and during the symposium itself. Anita Jones showed herself to be one of our typical faculty members, seizing all possible resources (faculty, staff, students) as the time of the symposium approached and putting it all together.

The technical quality of the symposium is reflected in these pages—the reader can judge for himself.

J. F. Traub

The Keynote Speech

Alan Perlis

Department of Computer Science
Yale University
New Haven, Connecticut

An after dinner speaker is usually classified as someone who is not otherwise gainfully employed, because it is really a full-time occupation to say something that will be interesting and not be buried in the occasion itself. And this is a very interesting and important occasion. I find that I am quite shocked and pleased at the large size of the turnout and the interest that so many of the alumni have shown in coming back to Carnegie.

I don't know why Carnegie is—if, indeed it is—different from other departments of computer science, though I suspect that Carnegie's role in shaping computer science is much stronger than anyone would guess from the size of the department, or the size of the school, or its location in Pittsburgh. I have no answer. I'll just lay out one suggestion. I think it's because the environment at Carnegie was not then, and I think is still not, rigidly stratified or organized into small cells of individually honed views of computer science. The department never consisted, and I don't think it does now, of a set of independent fiefdoms, where each professor rules supreme over his own piece of the computer science pie. Instead, there was always a large amount of communication and friction between people who held quite different views about what computer science was and ought to become. That difference existed then, and still does in the whole world, let alone Carnegie. It has had a lot to do with the freedom with which the graduate students, unencumbered by dogma when they left Carnegie, were able to maneuver in a new field. I hope that always remains true of computer science. A lot of other sciences have suffered badly from the fact that their feet became encased in cement much too soon.

I guess as an after dinner speaker it is important that I give an overview of computer science. I can't. Indeed, I think that most of the views that we get today are what I would call "underviews". I can think of very few blights that have befallen our field more disturbing than that called "structured programming". It has substituted preoccupation with style for concern with content. It has cast a pall on so much of what remains to be done. But it will wear itself out in time. Complexity theory is the latest entry in the mathematical sweepstakes. Refugees from automata theory and formal languages, cast out from their own countries have found a new grazing ground. However, they will soon denude it and computer science will go on its merry way again.

One of the great things about this field is its "robustness", to use a famous word of Edsger Dijkstra. It's very robust, and the reason it's robust is the computer. We keep finding new things to do with it, new groups of people who want to play with it, who bring new problems to it, and who turn to us for help. And each time they do that the field is revived. It takes off in another direction. Only an idiot, which I don't think I am, would say, "This that we do today is computer science, and will always be computer science."

Some years back in the publication that I hope my name will be bound most closely to, Herb Simon, Al Newell, and I submitted a letter to *Science Magazine* in which we said what computer science was. And in our confessed ignorance, we said it had to be the study of the phenomena arising around computers. We were right then, we're right today, and I think we'll be right twenty years from now. The computer should make us all humble because it is not possible for any of us, no matter how bright we are, or what our experience, to predict what it is going to be used for. It may ultimately disappear and be hidden under all kinds of gadgets and never be seen again, like the electric motor, and our only memories of the computer will be abstractions—things we talk about, things we draw, music we play, who knows? But to say that computer science is artificial intelligence, or complexity theory, or programming languages, or operating systems, or what-not, is ridiculous.

This makes our science, if we can call it that—and I think we should because it deals with phenomena—an extremely vital one. Departments that treat the subject that way flourish best. Both the faculty and the students approach each day open-eyed, open-minded, wondering what the day will bring in the way of new problems or new views. In a way this is also bad for many of us because it prevents us from digging very, very deeply. We don't generally tend in this field to dig very deep mines from which we disappear and are never heard from again. Those mines happen to be extremely attractive places in which to train people. Formal symbolic disciplines seem to be the ideal training grounds for students because you can train so many of them on a square foot of data.

Fortunately, this department, I think, has held to a course in which this has not been the *modus operandi*. Instead, there has been this very important open-eyed, young kid's view of the computer science world, which is constantly causing new things to be done and new points of view to be made. Certainly this is the way I've always viewed computer science and I think it's the way the faculty here has always viewed it. None of us are bright enough or wise enough to predict what this science is going to become. Anyone can predict what mathematics is going to become—more of the same. Physics—they're on a trail that will never end, searching for the ultimate. There's an aphorism of mine that says that "ultimate solutions immobilize". "The best is the enemy of the good", as Wittgenstein said. Be pragmatic. There is no such thing as truth in this field. There's only fun, the privilege to explore, to understand man.

I say the last thing very openly and without shame. I think it's extraordinarily important that we in computer science keep fun in computing. When it started out, it was an awful lot of fun. Of course, the paying customers got shafted every now and then, and after a while we began to take their complaints seriously. We began to feel as though we really were responsible for the successful, error-free, perfect use of these machines. I don't think we are. I think we're responsible for stretching them, setting them off in new directions, and keeping fun in the house. Fun comes in many ways. Fun comes in making a discovery, proving a theorem, writing a program, breaking a code. Whatever form or sense it comes in I hope the field of computer science, and this department in particular, never loses its sense of fun. Above all, I hope we don't become, or you don't—because it's bad enough when someone my age becomes one—and it's a disease of the 50s—I hope we don't become missionaries. Don't feel as if you're Bible salesmen. The world has too many of those already. What you know about computing other people will learn. Don't feel as though the key to successful computing is only in your hands. What's in your hands, I think and hope, is intelligence: the ability to see the machine as more than when you were first led up to it, that you can make it more.

Since I left Carnegie in 1971 and went to Yale I've been much concerned with how I can give students fun on the computer. What constitutes fun? When I say "fun", I don't mean playing Star Trek or chess. Those are other people's games. I mean solving puzzles, inventing algorithms, or becoming fluent in a language so that in a sense you know it through your entire body down to your fingertips. That's what I mean by fun. I've come to some interesting conclusions that may also be interesting to some of you. One is that ALGOL is a blight. You can't have fun with ALGOL. ALGOL is a code that now belongs in a plumber's union. It helps you design correct structures that don't collapse, but it doesn't have any fun in it. There are no pleasures in writing ALGOL programs. It's a labor of necessity, a preoccupation with the details of tedium.

LISP, I've found, does give a measure of fun, and the people who program in LISP really enjoy it. They have fun. The only other language that I know of that has this property is APL. And I have become a real APL nut. I find it is the first language that has anything remotely approaching the beauty of English, a language in which you can actually phrase things elegantly and 100 people will say things in 100 different ways, a language that converts the mind when you use it. So you begin to think in terms of patterns, idioms and phrases, and no longer pick up a trowel and some cement and lay things down brick by brick. The Great Wall, standing for centuries, is a monument. But building it must have been a bore.

When I participated in the creation of ALGOL, I felt very good. All of us did, because it was an awful lot better than FORTRAN. As the years have gone on though, I have come to realize—and I'm probably a voice in the wilderness here—that ALGOL really is a blight. In a sense, it is a language that belongs in the union, not on the playing fields of Eton. And as England learned over several centuries, all the important things were done on the playing fields of Eton and not on the scaffolds that went up around its buildings of state. I think the same thing is true of our field of work too. I like to think that if Shakespeare were alive today, he'd be a programmer and he'd be programming in APL.

Our country has lots of problems. One of them is BASIC. BASIC fits the minds of the people who run our secondary schools—simple, tidy, relatively cheap. And teachers find it is no more difficult to teach than anything else they teach. So it seems to fit into our secondary schools and kids are now coming to college already suffering from a disease that I don't think we'll ever be able to excise. When they come into my course, and I ask them to start programming in APL, I find they can't. They're crippled already. Their minds are already deranged. The first thing they do is set I equal to 1. The natural first step out of the womb is to set I equal to 1. Then you cast around for some place to use it. Let's use it in X, so you have $X(I)$. And you manipulate it for a while and then you increment I, and then you test and you jump, and that's programming. That's life.

When I tell these people that that's not the way to do things at all, but that they should look for a pattern and do things in parallel, to erect the whole structure at once, they can't. Then they come to me after a while and they say, "APL is unstructured. There's no *while* in it." I say, "It's got nothing but wiles", which pun they don't get. "There's no *for* statement; there's no *if–then–else*." I say, "It doesn't need them". I say, "What we do have in APL, what we have in English, what we have in all good natural languages, is a distribution of control through every part of the language, so that when one speaks or writes one weaves mosaics in which the context determines our meanings". That's what makes natural language good for people to use, and that's what been so sadly missing in programming languages. I try to tell these students something, things of that sort, and I'm afraid it's a long, lost battle, because they already know what programming is: *I* gets equal to 1.

However, fortunately, today three quarters or more of the students in high school don't learn programming, and they come to the university and they search around their schedule for an easy course, and we suck them in. These nonmathematically oriented people, people who are going to major in English, history, music, political science, etc., learn APL. For them it's not too difficult. After a couple of months when we show them FORTRAN or BASIC or ALGOL, they come back to us and say, "Why would anyone want to program in a language like that?" This is almost universally their attitude.

As a teacher, I am still interested in not digging deep holes, in teaching people fun. As I meet with graduate students in our department, I find that they have very serious questions to ask about our science. One of the things that bothers them is whether computer science is fenced in by some rather important natural barriers. I've given these natural barriers some rather fancy names. One, many of you know already—I call it the Turing Tarpit. The Turing Tarpit is a morass in which one, in a sense, finds that everything is possible and nothing is easy. One attempts to go from abstract models to real situations and finds that one cannot.

There's a second one that has recently come into being, and this is recognition that we are surrounded by mountains that are really unbelievably difficult or even impossible to scale. Indeed, the mountains are so bad that at the moment one of the greatest games around is to show that they are impossible to scale. All we're able to show is that one mountain is as bad as another one. These are tasks that are not impossible but exponentially time and space consuming. Everything of interest is outside of our reach. This bothers people—it bothers me I know—until I find out that the vast majority of the tasks that we do not yet know how to do are not in those categories.

There's one last "natural barrier", which is the most insidious of all. I call this the "semantic gulf". It's a large, placid body of water on which so many voyage. This gulf stretches out in every direction, and when we are upon it as a Columbus, we find that every time we reach a piece of land the continent we seek is farther away than it was before. Every time we learn something, we find that we knew less about what we were looking for than we thought we did. Every problem that we solve opens up a hundred new ones, each of which is more critical to solve than the one we just solved. This is not to say that we shouldn't voyage on this gulf, because I don't think we have any choice. It's in our nature that we take this voyage, but we must be aware of the fact that it's probably a voyage that we're unlikely to ever satisfactorily complete, hence a fantastic voyage, a fun voyage.

Nevertheless, I don't think all this makes our science weak; it makes it human. We have big goals in this science. Probably goals that are as big as any science, bigger than any science other than physics, or possibly molecular biology. Our goal is the complete and total understanding of the human mind. And we really believe that the computer will lead us to that goal. Well, when we get wise enough and humble enough we realize that we're probably not any closer to it now than we were when we started years ago. But it's been fun riding on the semantic gulf. It's a boat ride that's far more pleasurable than this one. You certainly don't have to listen to any after dinner speeches on it.

Well, I think I've said about all that I wanted to say. It's such a great pleasure to be back here. I think there are only a few faces that I didn't recognize and I can be forgiven because they've since become changed by growth. And I find the same spirit of fun, enjoyment in our work, that was in the department when I was here. Carnegie is still a great place for corrupting the sober and the serious, for turning them into human beings, not just automatons. So, in your next ten years, I wish you nothing but the best, and those of us outside of the department expect you to provide us with a new set of glorious, unbelievable, and useless inventions that will titillate our lives in the years to come. So let me wish you all "Bon Voyage" in the next so many years, and a good career to you all. Thank you.

What Have We Learned from the PDP–11?

C. Gordon Bell

Digital Equipment Corporation
Maynard, Massachusetts
and
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

In the six years that the PDP–11 has been on the market, more than 20,000 units in 10 different models have been sold. Although one of the original system design goals was a broad range of models, the actual range of 500 to 1 (in cost and memory size) has exceeded the design goals.

The PDP–11 was designed to be a small computer, yet its design has been successfully extended to high-performance models. This paper recollects the experience of designing the PDP–11, commenting on its success from the point of view of its goals, its use of technology, and on the people who designed, built and marketed it.

1. INTRODUCTION

A computer is not solely determined by its architecture; it reflects the technological, economic, and human aspects of the environment in which it was designed and built. Most of the non-architectural design factors lie outside the control of the designer: the availability and price of the basic electronic technology, the various government and industry rules and standards, the current and future market conditions. The finished computer is a product of the total design environment.

In this chapter, we reflect on the PDP–11: its goals, its architecture, its various implementations, and the people who designed it. We examine the design, beginning with the architectural specifications, and observe how it was affected by technology, by the development organization, the sales, application, and manufacturing organizations, and the nature of the final users. Figure 1 shows the various factors affecting the design of a computer. The lines indicate the primary flow of information for product behavior and specifications. The physical flow of materials is along nearly

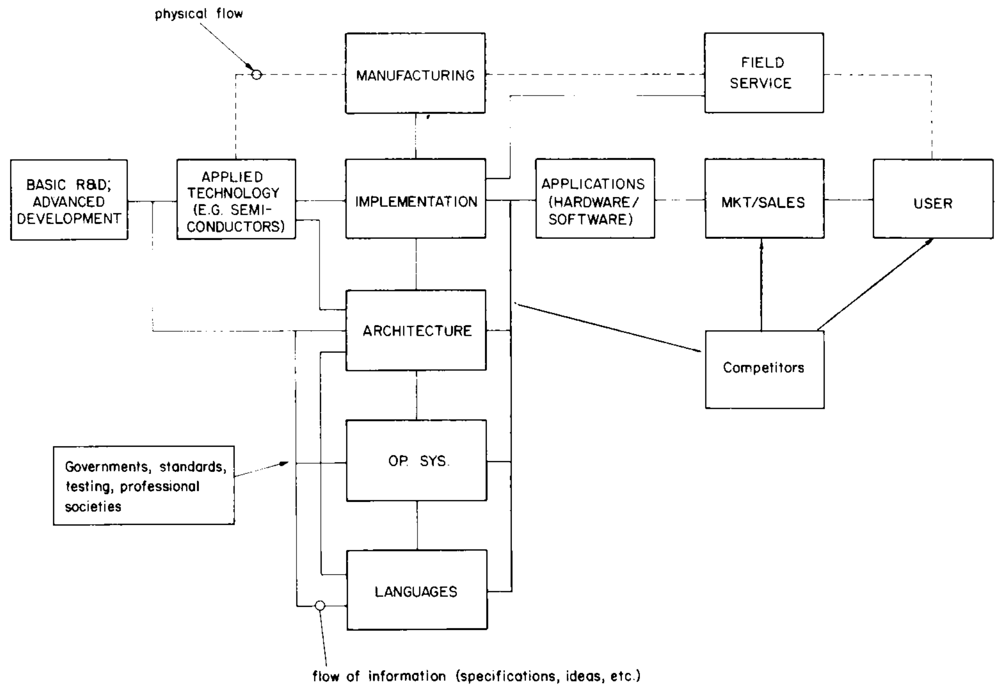
physical flow
MANUFACTURING
FIELD SERVICE
BASIC R&D; ADVANCED DEVELOPMENT
APPLIED TECHNOLOGY (E.G. SEMI-CONDUCTORS)
IMPLEMENTATION
APPLICATIONS (HARDWARE/ SOFTWARE)
MKT/SALES
USER
ARCHITECTURE
Competitors
OP. SYS.
Governments, standards, testing, professional societies
LANGUAGES
flow of information (specifications, ideas, etc.)

Fig. 1. Structure of organization affecting a computer design.