R E P O R T   R E S U M E S

ED 010 432 04
THE COMPUTER AND CAREER DECISIONS.
BY- ELLIS, ALLAN B. WETHERELL, CHARLES B.
HARVARD UNIV., CAMBRIDGE, MASS.
REPORT NUMBER BR-6-1819 PUB DATE SEP 66
REPORT NUMBER TM-1
GRANT OEG-1-061819-2240
EDRS PRICE MF-$0.09 HC-$0.80 20P.

DESCRIPTORS- *DECISION MAKING SKILLS, *SYSTEMS CONCEPTS, SYSTEMS DEVELOPMENT, *CAREER CHOICE, *INFORMATION SYSTEMS, *COMPUTER PROGRAMS, CAMBRIDGE, MASSACHUSETTS

THE NEED FOR STUDENT ACCESS TO A COMPUTER FACILITY, THE REASONING BEHIND THIS NEED, AND A GENERAL DESCRIPTION OF THE EQUIPMENT REQUIRED WAS REPORTED IN THIS TECHNICAL MEMORANDUM. THE NEED FOR THE DEVELOPMENT OF AN INFORMATION SYSTEM FOR CAREER CHOICE WAS PRESENTED. DISCUSSIONS OF RESEARCH INTENTIONS INCLUDED (1) A MODEL OF DECISION-MAKING, (2) THE BASIC COMPONENTS OF SYSTEM DEVELOPMENT, (3) RELEVANT COMPUTER PROGRAMS, (4) COMPUTER HARDWARE CONFIGURATIONS, AND (5) THE STAGES OF DEVELOPMENT ACTIVITY. (RS)

[illegible]

Technical Memorandum I

ED010432

THE COMPUTER AND CAREER DECISIONS

ALLAN B. ELLIS
***and* CHARLES S. WETHERELL**

Information System for Vocational Decisions
United States Office of Education
Grant No. 1–061819–2240

HARVARD UNIVERSITY
September, 1966

1 THE COMPUTER AND CAREER DECISIONS

This paper provides a fairly systematic statement of this project's need for access to a computer facility, the reasoning behind this need, and a general description of the equipment required. The paper is incomplete in two respects: it stops before providing either a specific list of the equipment needed or a technical evaluation of those computers on the market that might do the job; it speaks only about the major aspects of the theory on which the project rests, oversimplifying in some cases and disregarding ancillary or related aspects in others. It means to offer a general picture of our research intentions, which, in turn, will define the magnitude of our computer needs.

The purpose of this project is to develop an information system for career choice built on a paradigm of decision-making proposed by Tiedeman. This paradigm depicts the behavior of an individual from his earliest confrontation with a choice situation, to his final analysis of the results. In its simplest form, it states that there are two major components to decision-making—exploration and choice.

An individual, having recognized that there is a problem he must solve, enters the stage of *exploration*—sizing up alternatives and options; and locating risks and consequences—with the aim of structuring and ordering the universe of choices. The individual asks: What is available? What are the possibilities? How do they relate to my goals? What are my goals? How do the alternatives relate to each other? What is the universe from which I can reasonably select a course of action? Once these questions start to be answered, the individual is ready to consider choice.

One element in the stage of choice is the *moment* of choice—that instant at which commitment is made and begins to grow This moment is skirted by two phases, *crystallization* prior to choice, and *clarification* afterwards. Naturally, the actual selection of a course of action is important, but it takes on its fullest meaning only in terms of these two adjacent activities. In the crystallization phase of decision-making, the individual assesses the alternative paths of action. What are my chances? How have others like me managed? How much of a risk is really involved in each case? The products of exploration are worked over, pursued, refined, and sifted, until, finally, a selection is made. Then comes clarification, a more extensive following-through of the chosen path to see more fully what one has committed himself to, and to test how strongly he maintains his choice. Naturally, it is not a one-way road from exploration, to crystallization, to choice, to clarification, nor are they in fact as clearly separable as the paradigm might suggest.

Throughout the individual's passage from point to point in the decision-making process, he continues to engage in the act of turning data into information. This is a major concern of the project, since, in the real world, data are never complete. Often, it is precisely this incompleteness that makes decisions necessary in the first place. In any event, the quality of the choice depends upon the quality of the data. Before one attempts to make a decision, therefore, he must first understand the incompleteness of the data with which he is dealing.

Accepting data on these terms leads naturally to the condition that one is more likely to take responsibility for the choices he makes, since they are not totally determined by external factors. If they were, then choice would be either irrelevant or superfluous. Furthermore, in order to create information on which to base decision, one must *actively* process data rather than passively be guided by them, and therefore, the individual must become a significant agent in the choice process. That is, the incompleteness of data implies that the individual is *responsible* for his decisions in both meanings of the word: he is the one who makes the decisions, not someone or something external to him; he is the one who enjoys or suffers the consequences. This is one way to define 'freedom' and it is to this notion that the project is dedicated. It will achieve this goal by developing in the student the ability to engage in this kind of decision-making relative to his career choice. That is, the project will place the student among resources, enhance his access to them, teach him the stages in decision-making, and have him engage the resources in a controlled setting so that he can develop the skills of processing data and making decisions.

An additional factor in the decision-making procedure which this project proposes is called *monitoring* and consists in keeping track of the student as he goes from stage to stage through the paradigm. Aside from the usual reasons for monitoring a student's behavior—to analyze his performance, select from alternate courses of action, and generally maintain an account of his interaction with a system—the project expects to present to him the facts of this monitoring so that he might use them as additional data. These facts become a kind of meta-data which the student processes. The idea of data and meta-data is analogous to the philosophical notion of being and becoming. Not only does the individual act but he becomes aware of his pattern of action. The desired result is a higher order of understanding of both the decision-making act and the panorama of career choice in which decision points are linked. Career becomes a time-extended set of choices, and decision at any given point is enhanced by an overall awareness of the road being travelled.

What the project proposes, then, is a model of decision-making behavior which requires a setting capable of providing feedback. It is an interactive setting in which an individual engages a data base in certain specifiable ways as a means of determining alternatives and selecting from among them.

2

Based on the demands of this model of decision-making and on the complexities attending its implementation, we state that, to be successful, the project must develop its system on a computer.

One assertion that grows naturally out of this statement is that any attempt to specify the interactive model which this particular project offers must make reference to a computer. This assertion is not derived from a consideration of how a computer can assist in the development of an information system for career decisions. Such a consideration would imply that our concern for the role of the computer in the project stems from an interest in the computer itself. Rather, the assertion is derived from a consideration of the environment within which, by the very nature of our theory, we must place the student.

This setting we seek is one which will develop in the student the ability to reckon. The word *reckoning* carries many meanings: to count, figure up, compute; to take into consideration, regard, judge, estimate; a measuring of possibilities for the future; a calculated guess; the settlement of an account, or of rewards or penalties for any action; determination of a position (as a ship); to enumerate serially or separately; to place or class. We propose to use it in all its meanings, and to help the student transform himself into a reckoner. We will place the student in a reckoning environment and encourage him to take on more and more of the characteristics of this setting, that is, to become more and more a functioning element within it—in fact, its controller. This environment, therefore, must provide the student data with which to reckon, and it must be a functioning analogue of the ideal reckoner. To accomplish this, the environment must contain a reckoning device. It is clear that no such device exists at present that will totally satisfy these conditions, and as we will show later, the project cannot achieve its objectives without a close approximation to such a device. The computer—a machine for carrying out specified transformations on data—comes closest to possessing the desired characteristics and therefore is the device whose relevance will be demonstrated, not as a useful tool in conducting the project, but as the best candidate for the indispensable role as a functioning element in the interactive environment which our theory proposes. We will consider the role of the computer, then, in terms of data, decision-making, and monitoring.

Once we recognize the obvious fact that data are never complete, it becomes wise—often vital—to place the condition on choice that it be made with the best possible data available. We must ask of the data: Are they *accurate?* How *complete* are they? Do they reflect the full *complexity* with which we must deal? Can we get them *in time* to explore alternatives adequately? A library is unsatisfactory in this area, because the time involved in searching is often more than the individual can afford. Certainly large amounts of data—occupational descriptions, for example—can be stored, indexed, cross-referenced, and made generally available in a library, but that is only part of what is needed. The computer, on the other hand, is capable of all this and of providing fast access so that search time need not hamper decision-making. Furthermore, the computer can interact with the student and thereby help him to ask relevant questions about the world of work. The project looks to the computer, therefore, as a device to store large amounts of occupational data and to make them immediately and selectively available to the individual as he proceeds through the decision-making process. With this kind of accessibility, the individual can feel he is among resources and as he becomes more integrated into the reckoning environment, the data become more like extensions of him and less like external quantities.

The area of decision-making is the second one which the project sees to be dependent upon the use of a computer. As the student goes through the various stages of decision-making outlined in the paradigm, he must be provided a number of things, three of which are particularly important. First, he must have the opportunity to browse through a large set of sources and to obtain cues from the system as he goes. Browsing is not an aimless activity, it is an unstructured one, and the system should facilitate the emergence of order by probing, suggesting, reminding, and generally being at the disposal of the individual. Second, the student must be able to follow up leads, discard them, go as far as he wishes, and then start anew. There should be available to him statistical projections and estimates about himself and about the world of work with which he may compare his own interests and abilities to others who have taken certain courses of action, so that he can begin to assess his chances and his willingness to pay the price involved in particular alternatives. Finally, he needs a way to relive his past in terms of his present orientation and to act out the future or at least to bring his awareness of the future to bear upon his current need to make a career decision. The system, therefore, must contain some kind of simulator or game player which can process student information within the larger context of information about the world of work so that the student can engage in what amounts to a "career game."

All of this requires that there be an interface between the student and the data. In many respects a human being—counselor, librarian, teacher, friend, confidant—would be the best interface, although he would not be very good at treating variables in a multivariate way or at the kinds of real-time search, estimation, computation, recording and presentation that the system requires. It is in these areas that we look to a computer, since they involve operations which a computer can be expected to do well and tirelessly. This is not to say that one will replace the other; we have no such intention, especially since neither the counselor nor the computer is totally satisfactory. They will be used together to do what they do best and thereby to function in unison as the ideal reckoning component of the desired environment. The computer then is indispensable to our effort to create an environment in which the process of career decision-making is to be maximally facilitated.

A third area of our interest relates to a new interpretation of the role of monitoring. The idea of monitoring, of course, is not new, for many people have kept track of students' behavior for later analysis. We do not limit the role of monitoring, however, to diagnosis after the fact. We conceive of a real-time monitoring and feedback system where the student processes the data for monitoring on his way to a decision. The way a person engages a subject when he teaches it differs from the way he engages it when he studies. Not only does he think more about the structure and form of the subject as a means of explaining it, but, as a byproduct, he learns something more about the subject and from a different perspective. This kind of byproduct can attend the decision-making act as well, and we intend, therefore, to provide the student with information about the structure and form of his interaction with occupational data, and about his passage through the decision procedure, so that he can use it as grist for further milling.

There are, in particular, three things that our system wishes to provide the student relative to monitoring: the data for monitoring; the facility to reorganize his conception of the process; and the facility to analyze himself in terms of these data. At the end of a given session, the student might be asked to sum up his present situation, to state briefly the reasons behind his chosen course of action, and to describe how he anticipates things will go as a result of this decision. Some of this information might be obtained from tests or questionnaires administered by the system, while other information might be in-putted directly by the student. Given this situation, we can describe how the three monitoring functions might work.

The first monitoring function will be to store this set of information about the student for his use the next time he enters the system. When he re-enters the system, his first act might be to summarize how things have gone since his last meeting. Did he guess right? Did anything unusual happen? Was there any disappointment? Any surprises? The second monitoring function then will be to have the student compare his anticipation with what he thinks really happened, and to give him the facility to reorganize his present conception of the world in light of any disparities between prophesy and history. This activity is very much like that of essay writing where an individual comes back to an essay not only to *rewrite* it, but to *re-think* it as well. This re-thinking is based upon an awareness of the form and structure of his passage through time and as such, it represents a higher order of awareness. The goal is to get the individual to use this awareness to invent higher order concerns which are not dependent upon the *sequence* of history. Naturally, the results of this re-editing are to be stored and made available to the student at future encounters. The third capability, relative to the byproducts of the system's monitoring of the student, which he must be given, is that of analysis. Having produced what amounts to a re-edited version of his history, the student then needs to be able to look at it as a means of gaining greater understanding of himself. The function of the system is to provide the prompts to facilitate this understanding.

One approach to this might be to provide a content analysis 'package' where the student can examine his use of words, the sentences he has written and their juxtaposition, and those categories of word connotations which predominate in his work. In this way, not only does the student process a new kind of data and thereby gain greater insights into the paths he is taking, he also begins to deal with information about himself in a very personal and, hopefully, a more penetrating way. These demands of monitoring student behavior and of processing the products of such monitoring suggest that a computer is indispensable. At the very least, the time element involved in storing, retrieving, presenting, editing, updating, content analysing, restoring, cross-referencing and the like indicate that a human being would not be equal to the task. Again, we conclude that although a computer cannot be expected to carry the full burden, it is a vital component of the system which we envisage, both in terms of the monitoring capabilities just described and in terms of the more conventional functions of monitoring, such as the production of diagnostic data for analysis by a counselor or teacher, and the use of such data for decisions by the system at various branch points.

It is now clear that the role of the computer, with regard to data, decision-making, and monitoring, is crucial indeed. The answer to the question, "Why ever mention the word 'computer' in the same breath as 'career decision'?" is that, by its very nature, the interactive, reckoning environment, which this project postulates as desirable, requires functions to be performed which can only be performed by a computer and then only after considerable effort. It is the task of the project, then, to explore these functions in greater detail and to specify more precisely the interactive model and the things the computer is to do within it to enhance the process of career-choice.

In this way, we come to a second assertion which this project makes relative to a computer, namely that in order to develop these specifications of the interactive model, the project must have *access* to a computer. The question may be asked, "Why go through the trouble of becoming operational?" Our first answer is that by taking the model out of the realm of theory in an effort to provide an illustration, one gains verification. The model is too complex to be specified completely in the imagination. How well does it really work? What have we missed? Does it work for some people and not for others? Where are the malfunctions? Without a test in reality, the model can never be fully and properly specified. One anticipated by-product of this attempt to implement the theory is simplification. This is our second reason for stipulating the need for access. An attempt to make a functioning model brings a concern for all the working parts, and this attention to the details of implementation invariably suggests short cuts and more efficient procedures. We seek a prototype, therefore, to test the theory and to refine its implementation. Our third reason for wanting access is a strategic one, but of equal importance. With a working prototype, the system can have impact upon present practice. Without it, the project's influence upon these practices will be minimal. It is better to show the counselor that our system works than it is to have to convince him that it works. When we claim to be a developmental project, we mean exactly that our overriding goal is to *get into practice* the interactive model of career decision-making which we propose. While we accept the need to work out the various facets of the model, we recognize it as a prelude to implementation, and without entry into practice, the project becomes irrelevant to the pressing needs it is created to meet.

This is one way in which the theory and its implementation interact, though there is a more powerful and more continuing way. This more powerful relationship relates to the third assertion we make, that access to a computer is needed throughout the project's duration and not just at its culmination. By the very fact that the function of the computer in the model is to serve as its reckoning device, *the computer system becomes an analogue to the theoretical system being explored.* As such, development of the computer system facilitates development of the theory. We shall show further on that for each phase of the decision-making paradigm, there are particular computer software requirements. As computer programs are developed for each phase, the effect is that theory becomes recast in operational terms. The recasting is vital to the proper explication of the model since it places upon the paradigm the discipline and the formalism of algorithmic procedures. The proposed system will not spring into being full-blown. It will evolve stage by stage out of our increasing understanding of how theory and implementation relate and out of how well the implementation works in reality. It will succeed through a continuing interaction through the stages of theory$\leftrightarrows$program-logic$\leftrightarrows$reality test. Another important reason why continuing access to a computer is needed is that as students interact with the system, not only do we get a reality test, we also have the material needed to evolve along with the system, a diagnostic language for the man-system interaction we propose.

Simultaneous development of the model and the computer system to implement it is, therefore, most desirable. There would be the added feature in such an arrangement that even the first approximation to the facility we hope ultimately to provide would be better than what now exists, so that even as the system develops it can contribute to the field.

3

In its simplest form the computer-based information system for career decisions which this project will develop consists of three basic components: a student, a collection of data bases, and a set of computer programs which serve to interface the student and the data. Using the paradigm for decision-making described earlier as the basic expression of our theory, it is convenient to assume a relationship between the various stages of the paradigm and the functions which these computer programs are to serve. Naturally, the way a student interacts with the data base will vary depending upon whether he is in exploration, or crystallization, or clarification. Therefore, the functions performed by those computer programs which interface him to the data will necessarily vary as well.

As a first step toward defining the computer software requirements for the project, we propose that for the most part there is a one-to-one correspondence between the paradigm stages and the interface functions. This is the reason behind our earlier claim that the computer system becomes an analogue to the theoretical system being explored.

In the exploration stage, we see the primary character of the interface being that of a natural language question and answer system such as described in the works of Simmons, Bobrow, Cogswell and Estavan, Weizenbaum, Green, and others. That is, a system that can deal with the meaning of a question written in English rather than in some special language (numeric or program-like). The ideal system will not only answer such questions, it will also help the student form the questions, and thus it will be maximally interactive. The need for this comes, of course, from the fact that exploration is a highly unstructured activity. The student may not know what data he must deal with and the system must therefore be capable of probing, suggesting, reminding, and, in general, facilitating the student's access to the data base. The computer programs to be developed for this phase will (1) interact with the student to assist him in forming questions or expressing interests, (2) retrieve and present to him data relevant to these questions and interests, and (3) keep track of (monitor) the student's interaction with the system.

In crystallization, things become more structured, and the student's needs for information change. His concerns center more on risks, probabilities, the relationships of his abilities and interests to job characterization, and the like. At this point, the major characteristic of the interface will need to be that of a multivariate statistical calculator. Two principal functions of this statistical interface will be: (1) to process data about the student to locate him in some factor space or other reduced space, to locate him along some dimension as, for instance, a discriminant function, or to provide him with such things as expectancy tables, probabilities, and odds, and (2) to use the products of such computation as criteria for branches through the data base. An additional feature of this stage will be to present the student with certain tasks—such as completion of the Kelly Role Repertory Test, TAT, O'Hara's Vocational Self-concept Inventory, and Gribbons' Readiness for Vocational Planning test—in order to obtain further measurements on him for use in statistical computations and as added data for him to have at his disposal. The goal here is to make available to the student whatever data will assist him in following up alternatives and assessing possibilities on his way to choosing a course of action.

Clarification requires an interface with the characteristics of a simulator or game player. An individual enters clarification with certain questions on his mind: How are things likely to turn out? What have I committed myself to? Am I willing to pay the price? What are some of the future discontinuities (needs for decision) I must face because of my present choice? It is as though he had moved a chessman but has not yet removed his hand from the piece. He wants to take another, deeper look at what he has let himself in for. Perhaps he will change his mind, take back his move, and then sit back and explore the possibilities anew. These are some of the things the system must help him do by playing what amounts to a career game with him, using his 'chips', making moves on his behalf, using data from the student file and the data base to make the most educated guesses possible. The results of the simulation are used by the student to finalize his commitment and to form his expectations of how things will work out. These are the expectations he will re-examine at future encounters with the system as a means of getting greater insight into his choosing behavior.

Naturally, all of this is an oversimplification on two counts. First, the interfaces which we call question answerer, multivariate calculator, and simulator are neither completely separable, nor is each the sole function of its respective stage. Each stage in the computer system—as in the model—contains features of the others, and therefore each interface must possess similar capabilities with the difference being a matter of emphasis or power. Second, this is a simple description of the system and therefore does not reflect all the capabilities which the software must provide, such as for training students in the choice process, analysis of research data, counselor and teacher access to diagnostic data, and the like. Notwithstanding these qualifications, however, this description serves the present purpose of suggesting the nature and the magnitude of the software this project hopes to specify. Of course, our first efforts will be considerably more modest than these 'ideal' interface systems, but even these modest efforts will be in the direction of these ends. That is, the system will evolve its way toward these goals and therefore each cycle, as far as it goes, will be consistent with the theory upon which this project rests.

4

Our present effort has been directed toward searching the field to uncover existing, operating computer programs which might be relevant to our construction of these interfaces. In order to further suggest the nature of the programs we envisage as interacting with the student, and to provide a better feeling for both computer power needed and the amount of time involved, we present here a brief description of a few of these programs. Each program will be presented in a standard format, which is outlined below.

A. Program: The acronym of the program or set of programs and the translation of it.

B. Author: Either the group or the individual who developed the program.

C. Source: Either the place where we have been able to use the program, if it is running on a computer to which we have access, or the person or organization who has informed us of the program.

D. Language and machine: The usefulness of a program may depend on the machine-dependent features of it, as well as the language in which it is written. In an experimental situation, a translator to a machine to which we have access may not always be available.

E. Abstract: The author's abstract, if available; if not, one composed for this paper.

F. Implementation data: Although implementation will depend on the machine for which a program is being written, some general conclusions may be stated. Particularly, the relevance of section D may be commented upon.

G. Comments: Relevant information not contained in the above sections.

H. Bibliography: Our sources of information about the program, including especially titles of articles, manuals, and other written materials, and the names of the original authors or designers of the programs.

I

A. Program: TYPSET AND RUNOFF, an edit-by-context program and a right-justifying-type-out program.

B. Author: J. Saltzer, Project MAC at MIT, Cambridge, Mass., x-6039, MIT.

C. Source: *The CTSS User's Manual* and the Computation Center CTSS.

D. Language and machine: There is no information available about the language, but it is probably MAD and FAP. The machine is the MIT-modified 7094.

E. Abstract: From *The CTSS User's Manual.*
The command TYPSET is used to create and edit 12-bit BCD line-marked files. This command permits editing and revising by context, rather than by line number. The command RUNOFF will print out (in a format subject to control words placed in the file via TYPSET) a 12-bit BCD file in manuscript format. RUNOFF contains features which were not previously available, including type-justification.

F. Implementation data: Actual reconstruction of these programs would be unwise since RUNOFF is in very poor shape. However, these programs embody a large number of very good ideas, and are probably a better aid to manuscript creation than to a typewriter. Such programs are also the only feasible way to enter programs and hand-prepared data in a time-sharing system. Provided that it is economically feasible for secretaries to use the computer regularly, the six- months which it probably would take to implement a justifying and typesetting program (since justification is not as well understood as it might be) would probably be worthwhile, especially in conjunction with mechanical devices for preparation of photo-offset copy now available. A context editor is certainly necessary in any system such as ours to permit easy correction and updating of data, program and manuscript files.

G. Comments: There is a context editor at SDC which operates much like TYPSET, and there are several other context editors at MIT. The amount of work one can put into such a program is probably not limited by any consideration other than programmer time available. However, a right justifying program, such as RUNOFF, is something of a luxury. If much printing is to be done from typewriter masters, then this work may be worthwhile. A context and line editor for Harvard's IBM 360/50 is being prepared for another project by a member of ISVD and will be easily modifiable to ISVD's needs.

H. Bibliography:

The CTSS User's Manual, Sec. AH .9.01. Commands TYPSET and RUNOFF. Second ed., 1965.
——, Sec. AH. 3.01–.02. Commands INPUT, EDIT, FILE, TFILE, and ED. Second ed., 1965.
——, Sec. AH. 3.07. Command EDL. Second ed., 1965. Saltzer, J. MIT.
SDC Tech Memo. Command EDTXT.

II

A. Program: DSR, An Experimental On-Line Data Storage and Retrieval System.

B. Author: J. F. Nolan and A. W. Armenti, Lincoln Laboratory, MIT, Lexington, Massachusetts.

C. Source: The CTSS system and the authors.

D. Language and machine: No information, probably MAD, FAP, or LISP, or a combination of the three. The machine is the 7094 at Project MAC.

E. Abstract: From Lincoln Laboratory's report on DSR.

This report describes an experimental system designed to test and demonstrate on-line storage and retrieval of formatted data based on complete internal descriptions of files. The use of internal descriptions allows each user to define, modify, and cross-associate data files to suit his particular needs. The experimental program was implemented by remote use of the Compatible Time-Sharing System (CTSS) facilities of Project MAC at the Massachusetts Institute of Technology.

F. Implementation data: As with the LUCID system, to try to duplicate this program would be a waste of effort. But a combination of ideas from the two sources written especially for the hardware which we use would probably not be overly difficult, although a lengthy task. The structure of the programs is probably relatively simple. What is not simple is the programming needed to make these algorithms efficient.

G. Comments: Although we have not yet had a chance to run these programs, the write-up describes them as similar to the LUCID system at SDC. It is almost certain that this type of system will be needed in a general user environment. The programs require a rather rigid format for the internal description of data, but it is possible tha- this requirement may be relaxed so that inexperit enced users could use the system profitably.

H. Bibliography:

Armenti, A. W. and Nolan, J. F. An experimental on-line storage and retrieval system. Lexington, Massachusetts: Lincoln Laboratory, MIT, TM 377.
Armenti, A. W. DSR. An on-line data storage and retrieval system. Cambridge, Massachusetts: MIT, *User Memorandum* cc-263, MAC-M-314, CTSS.
Armenti, A. W. Lincoln Laboratory, x-7473.

III

A. Program: THE GENERAL INQUIRER.

B. Author: Philip Stone, Harvard University, Cambridge, Massachusetts.

C. Source: Allan Ellis and the Harvard University Computation Center.

D. Language and machine: The main routine is in COMIT and BALGOL for the 7094, with various input-output routines for display and retrieval of results written in AUTOCODER and SPS for the 1401.

E. Abstract:

This package of programs generates, from natural language textual input, a tagged output file. This file contains each sentence of the input and all relevant classes that each word of the sentence belongs to, as specified by a dictionary read by the program. These words and their tags can then be displayed in various combinations. The program is designed to aid critical reading of the text through revelation of attitudes and feelings, as evidenced by the relationships between dictionary classifications and the text.

F. Implementation data: The ideas behind this program are essentially simple ones and there should not be much difficulty in rewriting the package. However, because of the length of the programs, it will take at least several months.

G. Comments: This program and its uses will be discussed in a forthcoming book by Philip Stone, due to be published this winter.

H. Bibliography:

Stone, P. J., and Hunt, E. B. A computer approach to content analysis: studies using the General Inquirer system. *Proceedings, Spring Joint Computer Conference,* 1963.

STONE, P. J., and HUNT, E. B. The General Inquirer extended: automatic theme analysis using tree building procedures. Paper read at International Federation of Information Processing, Munich, Germany, 1962.

STONE, P. J., DUNPHY, D. C., SMITH, M. S., and OGILVIE, D. M. *The General Inquirer: a computer system for the analysis of text.* Cambridge: MIT Press, in press.

IV

A. Program: PROTRAN II.

B. Author: Carl E. Helm, Princeton University, Princeton, New Jersey.

C. Source: Allan B. Ellis.

D. Language and machine: FORTRAN for the 7094.

E. Abstract: From *Simulation Models for Psychometric theories*, Carl E. Helm.

This project is directed to the exploration of the procedures used by psychologists in evaluating a person's performance on a battery of psychological tests or scales. During the course of this research a language was developed and programmed for the 7094 which greatly facilitates the development of test interpretation systems. The basic element of this language is the "sentence-generating operator" which relates a class of verbal material to a pattern of scores. Any operator may appear as an argument of any other operator, making it possible to represent complex decision rules in a natural way. Moreover, the class of verbal material defined by any operator may occur as an element in the verbal material defined by any other operator, making it possible to develop complex linguistic structures as well. The system is being successfully used to produce clinical interpretation of complex profiles of test scores.

F. Implementation data: Not available.

G. Comments: We have a listing of the program and Carl Helm is interested in working with us in time-sharing it.

H. Bibliography:

HELM, CARL E. Simulation models for psychometric theories. Princeton University, November, 1965.

HELM, CARL E. Computer simulation techniques for research on guidance problems. Paper presented at the annual meeting of the American Personnel and Guidance Association. April 5, 1966.

V

A. Program: ELIZA.

B. Author: Prof. Joseph Weizenbaum, MIT, Cambridge, Massachusetts.

C. Source: CACM, January, 1966.

D. Language and machine: This program is written in MAD-SLIP, MAD, and FAP. It runs on the 7094.

E. Abstract: From CACM article, January, 1966.

ELIZA is a program operating within the MAC time-sharing system at MIT which makes certain kinds of language conversation between man and computer possible. Input sentences are analyzed on the basis of decomposition rules which are triggered by key words appearing in the input text. Responses are generated by reassembly rules associated with selected decomposition rules. The fundamental technical problems with which ELIZA is concerned are: (1) the identification of key words; (2) the discovery of minimal context; (3) the choice of appropriate transformations; (4) generation of responses in the absence of key words; and (5) the provision of an editing capability for ELIZA "scripts."

F. Implementation data: The entire philosophy of this program is determined by the author's use of his list processing language, SLIP. However, the present implementation of this language is highly machine-dependent. The program is also being expanded in some rather complex directions. To implement this system will be a major effort, although a subset of the full system could be done fairly quickly.

G. Comments: For a complete technical discussion, see the forthcoming paper by David Archibald. Presently, there are some devices, mainly used for decision, that are not written up in CACM. Script writing is definitely a complicated task that demands a great deal of understanding of both psychology and the machine.

It may also happen that SLIP is not the best way to implement the list processing aspects of ELIZA, especially when the program is moved to a machine other than the 7094. For example, AMBIT seems to bear promise in this area. ELIZA is becoming very similar to a higher level programming language and this suggests a compiler approach. More study must be done on this problem.

H. Bibliography:

ARCHIBALD, DAVID. Internal technical memorandum on ELIZA, unpublished.

WEIZENBAUM, JOSEPH. Symmetric list processor. *Communications of the ACM*, September 1963, 6, 524–544.

——. ELIZA—a computer program for the study of natural language: communication between man and machine. *Communications of the ACM*, January, 1966, 9, 36–45.

VI

A. Program: LUCID (a system consisting of LUTRAN, LOADER, QUUP, ORNZ, SYMOUT, and MERG).

B. Author: Mr. Al Vorhaus, SDC, Santa Monica.

C. Source: SDC time-sharing system.

D. Language and machine: These programs are written in JTS JOVIAL for the Q-32.

E. Abstract:

This package is used to describe, build, query, sort, merge together, and update data-bases of relatively fixed format. The description phase consists of the specification of format for the generalized elements of the data base. Loading is done by a program which translates the descriptions and applies them to input data of a standard format (thus enabling the generation of more than one data base from the same data). The query program allows the user to sort the elements of the data base and then retrieve all fields which fit some specific criterion. All of the work is done on-line.

F. Implementation data: As is the case with the other SDC programs, these are written in a non-standard language with the emphasis on operation rather than clarity. However, the basic ideas are clear. Implementation would be a long job, however, and the final set of programs would be a large complex.

G. Comments: This is the kind of program that would work best in retrieving structured information, such as that being generated by Davis's group. There are, however, other such programs now available. Combined with DSR, for example, the ideas in LUCID could be used to generate a very powerful tool for data handling.

H. Bibliography:

MERG User's Manual. Santa Monica: SDC TM 2880/000/02.

ORNZ User's Manual. Santa Monica: SDC TM 2826.

QUUP User's Manual. Santa Monica: SDC TM 2710/000/01.

SYMOUT User's Manual. Santa Monica: SDC TM 2920/000/00.

TSS-LUCID LOADER User's Manual. Santa Monica: SDC TM 2936.

The TSS-LUCID Reactive Translator (LUTRAN) User's Manual. Santa Monica: SDC TM 2844.

VII

A. Program: F-2-STAT or F-4-STAT

B. Author: Albert Beaton, Educational Testing Service, Princeton, New Jersey.

C. Source: Albert Beaton.

D. Language and machine: 7094 FORTRAN.

E. Abstract:

This is a set of matrix operators, about eight in number, that may be combined to generate almost any statistic. The operators mirror in practice the simplicity of the linear algebra approach to mathematics. They are simple to combine and use.

F. Implementation data: Implementation should be fairly easy, since the coding comprises only about 250 lines of FORTRAN. More to the point, however, is the provision for an environment in which these routines can live. Simply coded as subroutines, they are of little value, since means of combining exist only by writing a separate main program for each task. However, a larger environment program could remedy this situation.

G. Comments: There are a number of statistical packages extant today. However, most of them provide a separate routine for every statistic desired. This package is simpler in concept and smaller in size, while providing much greater flexibility than other statistical systems.

H. Bibliography:

BEATON, ALBERT. The use of special matrix operators in statistical calculus. *Research Bulletin*, Princeton, New Jersey: Educational Testing Service, 1964.

LABRIE, RICHARD. The feasibility of providing statistical power to a text handling language in a computer-based educational system. Unpublished paper, 1966.

VIII

A. Program: TRAC, Text Reckoning and Compiling.

B. Author: Calvin Mooers, Rockford Research, Cambridge, Mass.

C. Source: Calvin Mooers.

D. Language and machine: PDP-1 assembly language, GE Datanet-30 Assembly language, ALGOL, and MAD.

E. Abstract: From CACM article, March, 1966. The TRAC language was developed as the basis of a software package for the reactive typewriter. In the TRAC language, one can write procedures for accepting, naming and storing any character string for the typewriter; for modifying any string in any way; for treating any string at any time as an executable procedure, or as a name, or as text; and for printing out any string. The TRAC language is based upon an extension and generalization to character strings of the programming concept of the "macro." Through the ability of TRAC to accept and store definitions of procedures, the capabilities of the language can be indefinitely extended. TRAC can handle iterative and recursive procedures, and can deal with character strings, integers, and Boolean vector variables.

F. Implementation data: Implementation of TRAC is relatively easy. It is a simple and highly modular program. The standard ALGOL description, which is now in the process of being debugged, will aid further in this field. The program is also a very small one, probably not taking more than two or three thousand words of storage on any machine.

G. Comments: TRAC is a string-handling language and is in the same line of development as SLIP, LISP, COMIT, SNOBOL, IPL, or any of the other string and list languages now in use. While TRAC itself has not been demonstrated to be valuable in a practical way, these other languages have, and TRAC seems to have advantages over some of them. It is certainly true that the intellectual stream to which TRAC belongs has brought some major developments and is causing a great deal of new and ingenious work to be done. TRAC also resembles very closely Strachey's General Purpose Macrogenerator (GPM), the theoretical model of all present macro languages. This also would tend to indicate that it is a valuable language.

H. Bibliography:

The CTSS User's Manual. Command GPM.

Mooers, Calvin. An ALGOL definition of TRAC. Program in preparation.

——. TRAC, a procedure-describing language for the reactive typewriter. CACM, 1966, 9, 3, 215–219.

——. TRAC, a text handling language. *Procedures of the ACM 20th National Conference, Cleveland,* August, 1965, 229–246.

Strachey, Christopher. A general purpose macrogenerator. *Comput. J.,* 1966, 8, 3.

IX

A. Program: AN ALGEBRAIC DESK CALCULATOR

B. Author: Many authors at various places.

C. Source: SDC, MIT, and others.

D. Language and machine: Not applicable.

E. Abstract:

An algebraic desk calculator is a program which will cause the computer to act like a very sophisticated calculating machine. Besides the ordinary arithmetic calculations, there must be the provision for the common functions and for the definition of new functions and procedures by the user. Variables can be defined and stored.

F. Implementation data: Implementation time is hard to estimate. A program like this is always open to the addition of new facilities. However, several months would be needed for a prototype.

G. Comments: Such a program is a great aid to investigators in doing short calculations. But implementation could well be deferred until more important projects are in hand. One finds, however, that a tool of this sort quickly becomes indispensable, once it is put into use.

H. Bibliography:

CTSS User's Manual, Sec. AH. 2.18. Command FORMAC:

SDC Tech Memo, Command EVAL.

SDC Tech Memo, Command PLANIT.

X

A. Program: PLANIT, Programming Language for Interactive Teaching.

B. Author: Mr. Sam Feingold, SDC, Santa Monica.

C. Source: The SDC Time-Sharing System.

D. Language and machine: The program is written in JTS JOVIAL for the SDC Q-32.

E. Abstract:

PLANIT is a general purpose teaching system which allows a lesson designer to enter course content into the computer for use as a teaching device. PLANIT operates on the SDC Q-32 time-sharing system. The user (i.e., lesson designer or student) communicates with the system via a keyboard device which is linked to the computer by either telex or telephone. Interacting with PLANIT, the user can build and edit lessons, present lessons and perform computations.

F. Implementation data: PLANIT is in poor shape at this time. The programmers have been more concerned with getting the system running than with good design. There are a large number of separate devices all combined into one large program in this package. To try to duplicate it would be folly. However, the ideas included in the system are usually rather good, and it would take less effort to generate them in a well-designed system of our own. Six months would still be a minimum to get a program of this size and complexity running.

G. Comments: This program, while large and complex, is relatively easy to learn to use for simple lessons. Lessons which do much branching or problem presentation require more effort. The package includes a phonetic encoder, a keyword searching device for the extraction of keywords from lengthier answers, and a computation facility of a very high order, rivaling most on-line desk calculating simulators. It also includes certain built-in programming capability, and the ability to use programs and lessons other than PLANIT and the lesson being used at the time of the branch. A full write-up appears in the SDC Tech Memo.

H. Bibliography:

FEINGOLD, S. L. *Users Guide to PLANIT.* Santa Monica: System Development Corp., TM-3055/000/00.

XI

A. Program: AUTOCOUNSELOR

B. Authors: Donald Estavan and John F. Cogswell, SDC, Santa Monica, California.

C. Source: The SDC Time Sharing System.

D. Language and machine: SDC JOVIAL for the Q-32.

E. Abstract:

This program is an attempt to guide students in their choice of courses throughout their high school careers. By asking the student to consider the required courses for the school and the type of further education or vocation which the student plans to enter, the program induces the student to look more closely at his four year program. There is also a segment of the program which compares the student with others having his same grade profile and career plans.

F. Implementation data: This program is useful only as a demonstration of the power of the computer to retrieve and handle records in response to human situations, and does not show enough sophistication to be worth implementation.

G. Comments: Section F essentially covers the situation.

H. Bibliography:

COGSWELL, JOHN F. and ESTAVAN, DONALD P. Explorations in computer-assisted counseling. Santa Monica: System Development Corporation, 1965, TM-2582/000/00.

XII

A. Program: SSP for Scientific Subroutine Package.

B. Author: IBM Application Program Division.

C. Source: IBM.

D. Language and machine: FORTRAN for the IBM System 360.

E. Abstract: From the SSP Programmer's Manual.

The Scientific Subroutine Package (SSP) is a collection of over 100 FORTRAN subroutines divided, for the sake of presentation, into three groups: statistics, matrix manipulation, and other mathematics. It is a collection of input/output-free computational building blocks that can be combined with a user's input, output, or computational routines to meet his needs. The package can be applied to the solution of many problems in industry, science, and engineering.

F. Implementation data: Since these subroutines are written in standard FORTRAN, there would be little trouble getting them to run in almost any environment, except where problems of rounding error obtrude.

G. Comments: This approach to the problem of providing mathematical, and in particular, statistical power to the programmer and user, is in direct contradiction to that of Beaton in the F-2-STAT and F-4-STAT packages. Here there is a specialized program for each detailed problem. It may be argued that this increases program efficiency, but it is likely to cost much more in time spent by the user trying to find precisely the right program for his problem. The relative merits of the two approaches must be evaluated in terms of the users and the environments in which the systems will be used.

H. Bibliography:

System/360 scientific subroutine package (360-CM-03X). *Programmer's Manual.* White Plains, New York: IBM Technical Publications Department, 1966.

System/360 scientific subroutine package (360-CM-03X). *Application description.* White Plains, New York: IBM Technical Publications Department, 1966.

XIII

A. Program: META-5.

B. Authors: David K. Oppenheim, Abacus Programming, Los Angeles, California, and Daniel P. Haggerty, SDC, Santa Monica, California.

C. Source: The SDC Time Sharing System.

D. Language and machine: SDC JOVIAL for the Q-32.

E. Abstract: From the SDC META-5 Manual. This language can be used to describe the syntax of formal languages and the transformation from one formal language to another. The system can then be used to take these rules of syntax and transformation and execute them, i.e., operate them on a computer.

F. Implementation data: This program, in its present state, would not be one which we could use. But the ideas could be implemented in another program.

G. Comments: Although the author's ideas about this program are couched in the terminology of formal languages (that is, languages such as the very strict programming languages), it should be noted that these same techniques can be applied, after some conceptual twisting, to both natural languages and data description. ELIZA performs this same type of transformation in a more heuristic and intuitive framework. Similarly, DSR and LUCID can be thought of as programs that transform raw data into other formal structures. These ideas are also valuable in the building of new compilers for special purpose languages as the project needs them.

H. Bibliography:

OPPENHEIM, DAVID K. The META-5 language and system. Santa Monica: SDC, 1966, TM 2396/000/01.

—— and HAGGERTY, DANIEL P. META-5: a tool to manipulate strings of data. *Procedures of the ACM 21st National Conference, Washington,* 1966, 465–468.

XIV

A. Program: MSA, for The Multivariate Statistical Analyzer.

B. Author: Kenneth J. Jones, Harvard University, Cambridge, Massachusetts.

C. Source: Kenneth Jones.

D. Language and machine: FORTRAN for 7094.

E. Abstract:
This set of programs is much like SSP, differing only in that there is more emphasis on statistics and matrix calculations and none on mathematical routines.

F. Implementation data: Since these routines are in FORTRAN, they should operate in almost any environment, except where problems of rounding errors obtrude.

G. Comments: See SSP.

H. Bibliography:

JONES, KENNETH J. The multivariate statistical analyzer. Cambridge, Mass., 1964.

XV

A. Program: SOUNDEX coding.

B. Author: None.

C. Source: Honeywell Corporation.

D. Language and machine: Not applicable.

E. Abstract:
SOUNDEX coding is a method of coding words into one common phonetic equivalent. Thus, "George" and "Jorg" might both reduce to "grg" under the process. It is most commonly used to handle misspellings of a phonetic nature. Interactive programs such as ELIZA and QUEST might use this method with profit. There is already a version of it in PLANIT.

F. Implementation data: Not applicable.

G. Comments: Any interactive device must be prepared to handle errors by users gracefully. Perhaps the most common type of error which we may expect to encounter is the spelling error or the typing error. This technique could handle the problem in a large number of cases. It is already widely used in the insurance business, where variant spellings of the same name are common. No one seems to know where the idea originated, but Honeywell sells a package of programs employing the method to the insurance business; the PLANIT programmers have taken over a version of it designed by Patrick Suppes at Stanford; and Remington Rand markets a clerical aid for the use of the technique known as SOUNDEX cards. The principle is not difficult to work out in detail.

H. Bibliography: None.

5

This project proposes the development of an interactive reckoning environment in which an individual's exercise of a career decision-making procedure will be maximally facilitated. Within this environment, the computer will be used as an approximation to that reckoning device which our theory calls for. Essentially, the computer will be used (1) to store large amounts of vocationally relevant data, such as occupational descriptions, job characteristics, statistical projections of manpower needs, and student data files; and (2) to provide access to these data on demand or as a result of certain interactions with the individual. We now consider what this computer should be like in general, and some of the specific things it must be able to do.

At once we draw the obvious conclusion that such a computer should be able to communicate with at least one individual at a time. We mean by this the kind of communication one might have at a teletypewriter or inquiry station connected to a computer. Accepting for the moment the restriction that the computer accommodate only one person at a sitting—that is, that there be only one station "live" at any time and that until the person using that station signs off, no other person can use the system—it is clear, nonetheless, that there will need to be more than one station connected to the computer. It is unreasonable to require all individuals—programmers, researchers, students, counselors, teachers, and the others to whom the system will be made available—to come to a single location to gain access to the system, particularly since it is our intention to provide our model as an integrated part of the various environments in which these individuals function. We conclude then, that the computer should possess the capability of being fed information from a number of remote-inquiry stations. This is what is ordinarily meant by "multiaccess computer." Considering next the restriction that the computer accommodate only one person at a sitting: we find this untenable. The trouble of setting up time schedules, allotting individuals prearranged amounts of time, rescheduling because of changes of plans or unforeseen circumstances, and the like, would be reason enough for us to reject such a condition. More important than this, however, is the fact that since no individual could ever use the full capacity of the computer for the entire duration of his communication with it, this restriction becomes indefensibly wasteful. Finally, of course, it is inconsistent with the purposes of this project that a person could not have access to the system without excessive delay. It becomes desirable then to require that more than one individual be given access to the computer at what amounts to the same time. That is, that more than one remote-inquiry station —in fact, most or all of them—should function as though they were "live" at the same time. The popular term for this condition is "time-sharing" (although "time-slicing" is probably less misleading)—a technique of allotting computer time to different remote stations in little quanta or slices so minute when compared to human reaction time that each user has the illusion of having the machine to himself. Given that the computer will "time share," we make one final conclusion—that the computer permit multiprogramming. This term is used to describe a system in which different programs, or more precisely parts of different programs, reside in the main computer memory at the same time. The usefulness of multiprogramming is in incorporating the most efficient use of the hardware resources into an operating system. Often programs waste time waiting for some input/output device to be in a state where it can be used. The multiprogrammed operating system utilizes this time by switching control to another program until the delaying program is ready to resume operation. This conclusion regarding multiprogramming stems from the desire to keep to a minimum the amount of time that a student, on the average, will have to wait for a response from the system. We offer, therefore, the general statement that this project requires a multiprogrammed, multiaccess, time-shared computer.

We now offer some specific statements regarding the computer hardware configuration needed by the project. We consider this configuration "generic" in the sense that not every element is specified and in that there is latitude in certain categories. However, it represents in general what we believe to be a minimal configuration, capable of meeting the demands of our project. Whenever it is necessary to make reference to some specific component, IBM equipment will be used. They will serve as examples only and do not necessarily represent our feelings as to which vendor's product is the best.

Core

Using, as a point of reference, the operating system 360—that set of software designed to simplify such housekeeping as input/output procedures, data and job management, and the like, and thereby facilitate our development of a multiprogrammed, time-shared system—a typical (but not necessarily adequate) operating system could permanently occupy some 52,000 bytes of main storage.* (Reference: IBM OS Manual on Storage Estimates—example 4). Since this includes neither user-written programs nor work space, it seems clear that 65,000 bytes are insufficient. This suggests that a 360/30, for example, would be inadequate in this respect, although in other respects it might be satisfactory. Given the functions that the "interfaces" are to perform—natural language question answering, multivariate statistical calculating, and game playing—it appears that a core size of approximately 130K bytes (in general, the next larger sized core) would be a minimum. This would make the 360/40 with a core capacity ranging up to 256,000 bytes, the smallest machine in the IBM line which could be used.

Central Processor

The central processor must be capable of handling many communication devices and large amounts of random-access auxiliary storage. In addition, it must contain those extra hardware facilities needed to support (1) time sharing (e.g. memory protect, interrupt, indexing) and (2) reasonable scientific computation (floating point option, etc.).

Auxiliary Storage

The project requires large amounts of random access auxiliary storage for data files, student files, occupational description, and the like. Naturally, the store may be graded with the fastest access devices being used to support the time-sharing system. Our present estimate is in the vicinity of 50 million bytes, although this figure is not a firm one at this time.

Terminals

The two points we consider here concern the number of terminals needed by the project and the particular capabilities they must have.

We estimate that six terminals would suffice for the first phases of the project, after which about fifteen terminals would be required. At first, these terminals might be located at the computer site, and later moved out to such a remote location as the Newton Public Schools.

Naturally, these terminals would need to be equipped with an input/output device. Furthermore, since the project is concerned with the effectiveness of various media of presentation and is interested in providing maximum variety, these terminals ought to have audio and visual (film strip and CRT) capabilities, as well as a keyboard/printer. The IBM 1500 Instructional Station comes closest to meeting these requirements.

Basic Equipment

We add to these requirements the basic equipment needed at any computer to support operating systems: card reader and punch; line printer; four tape units; operator's console.

Although this description is general, it reflects our needs sufficiently well to provide the limits

* An adequate FORTRAN compiler added to the system, for example, would require 256,000 bytes.

within which the computer used by the project must reside. Given that the need for a computer of this generic configuration has been established, one area that must be considered relates to the alternate arrangements possible to provide this project access to such a computer.

6

In developing the computer system for the project, there are three successive stages of activity in which we will engage.

The first is *survey*. Typical activities of this stage are locating computer programs which seem to be useful to us, trying them out, assessing their relevance, conceiving of changes that would make them more suited to our purposes, and studying the various algorithms that these programs use. A certain amount of programming on our part will be necessary. But, since a large majority of these programs in which we are interested have been written on general purpose time-sharing systems—such as ELIZA at M.I.T. and QUEST, PLANIT, and AUTOCOUNSELOR at S.D.C.—the most desirable arrangement during this phase would be to have telephone connection to the computer in question.

The second stage is *testing*. This consists of a more extensive application of these programs—using them on our own data, making changes in them, linking some of the programs together, and the like. In addition, we will begin writing our own programs and developing the operating system for the project. Even though use of general purpose time-sharing services is by far the most desirable arrangement during the survey stage, it is inadequate for testing, since it is not possible, in general, to be allotted sufficient storage to provide access to a large data base of the kind envisaged by the project. Furthermore, it would not be possible to have very much control over the computer, even to the extent—in the case of M.I.T.—of getting tapes mounted when needed. S.D.C.'s system is by far the best for our purposes, yet it provides inadequate, often sporadic, access. The remoteness of S.D.C. (Santa Monica, California) and the telephone costs argue severely against even this system's being anything but an expensive nuisance. The alternative of purchasing time on a general purpose time-sharing service, therefore, does not seem to offer the flexibility of access required at this stage. During testing, we require the kind of access that would be provided by our own computer, with the exception that since this is not an operational phase, we would not make full use of the computer enough of the time to justify the cost. The preferred arrangement, therefore, would be to have access to a computer we might treat as our own for a predetermined amount of time each day. Naturally, this would be quite satisfactory, although there are, of course, difficulties with such an arrangement, including the possibility that our conditions of access and control might not be reasonable from the standpoint of the person from whom we purchase such time. This might not be a problem, however, if the "outside" computer, in fact, belongs to the Harvard Computing Center, where, as members of the same family, we might be given the extra consideration that this alternative requires.

The final stage of activity is *implementation*. In this phase we will be treating the system as operational, although, of course, not as the final product. It is here that the project will be making its maximum use of the computer—writing, debugging, and testing programs, running subjects through the system, performing statistical analyses on derived data, and providing counselors and teachers with access to the data base. Not only will our needs for access be greatest during this time, but they will have to be met on demand. This is the point at which the project must get its own computer.