1965

Summary Report

AUGMENTING HUMAN INTELLECT: EXPERIMENTS, CONCEPTS, AND POSSIBILITIES

By: D. C. ENGELBART

Prepared for:

DIRECTORATE OF INFORMATION SCIENCES
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
WASHINGTON, 25, D.C.

CONTRACT AF 49(638)-1024

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

March 1965

Summary Report

AUGMENTING HUMAN INTELLECT: EXPERIMENTS, CONCEPTS, AND POSSIBILITIES

Prepared for:

DIRECTORATE OF INFORMATION SCIENCES
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
WASHINGTON 25, D.C.

CONTRACT AF 49(638)-1024

By: D. C. ENGELBART

SRI Project 3578

Approved: TORBEN MEISLING, MANAGER
SYSTEMS ENGINEERING LABORATORY

J. D. NOE, EXECUTIVE DIRECTOR
ENGINEERING SCIENCES AND INDUSTRIAL DEVELOPMENT

ABSTRACT

Early stages of research on augmenting human intellect are reported. An experimental system comprising a tapewriter (tape-punching typewriter), a computer-programmed Translator, and a Flexowriter and incorporating a text-editing facility has been implemented and was used in composing much of the original rough draft of the report. The Z-code system that provides the editing operations is described.

The general philosophy of the system is described, and then an illustration is given in which the philosophy is applied to a particular problem. A chord handset is discussed that will permit one or two-handed binary transmission, and possible results of its incorporation into the system are mentioned.

Questions still to be answered and speculations about possible future results are included.

FOREWORD

This report describes the AFOSR-sponsored portion of a study being carried on at Stanford Research Institute, under Contract AF 49(638)-1024. Other portions of the study are being supported by ARPA under Contract SD-269, by AFSO under Contract AF 19(628)-4088, and by NASA under Contract Nas 1-3988.

The project monitor for the AFOSR portion is Mrs. Rowena Swanson.

CONTENTS

CONTENTS (continued)

ILLUSTRATIONS

I INTRODUCTION

This is the second report[1,2] on a continued effort whose nature demands some introduction. "Augmenting human intellect" here means increasing the capability of a man to approach a complex problem situation, to gain the comprehension that he seeks, and to derive solutions. Increased capability in this respect is taken to mean one or more of the following: faster comprehension, better comprehension, comprehension of the previously incomprehensible, speedier problem solving, better solutions, or solutions to previously insoluble problems.

None of this is a matter of clever tricks for particular situations; instead, an integrated domain is envisioned where hunches, cut-and-try, intangibles, and the human "feel" for a situation coexist usefully with powerful concepts, streamlined terminology and notation, sophisticated methods, and high-powered electronic aids. The subject matter envisioned is the level of complex problem faced by scientists, diplomats, attorneys, and executives. The means of augmentation are extensions of those man has already developed to assist in applying his native sensory, mental, and motor capabilities.

Along the conceptual and planning lines reported in References 1 and 2, an experimental program was launched with separate, but coordinated, projects supported by ARPA, ESD, and NASA. These projects have pushed the development of on-line text-manipulation techniques that are presently being integrated into our way of work.

---

[1] Engelbart, D.C., "Augmenting Human Intellect; A Conceptual Framework," Contract AF 49(638)-1024, Stanford Research Institute, Menlo Park, California (October 1962), AD 289565

[2] Howerton, Paul W. and David C. Weeks, Vistas in Information Handling, Vol. I, Douglas C. Engelbart, "A Conceptual Framework for the Augmentation of Man's Intellect," Pgs. 1-29, (1963), Cleaver-Hume Press

The smaller AFOSR project, the precursor of the others, has provided conceptual and planning guidance, and has pursued several experimental developments associated with the program objectives. This report deals only with work of the AFOSR project. Reports on the work of the other projects are planned by mid 1965.

The system approach to increasing human intellectual effectiveness finds no ready-made conceptual framework such as exists for established disciplines. Thus, before a research program could be designed intelligently, a conceptual framework had to be erected. Briefly, the conceptual framework has the following basis: The intellectual effectiveness of a human can be significantly improved by an engineering-like approach toward redesigning changeable components of a system. The principal elements of the system involved here are the language, artifacts, and methodology that a human has learned to use. These elements are dynamically interdependent within an operating system whose structure is hierarchical. That is, there is a hierarchy of process capabilities whose primitive components are the basic human capabilities and the functional capabilities of the artifacts, organized into successively more sophisticated capabilities. An example of this is discussed in Chapter II, in which the various human capabilities and artifact capabilities concerned in man-to-computer communication are discussed, and a possible new solution is presented--a chording handset that may permit of great potential improvement in direct, man-to-computer "speech."

The continued work at the conceptual-framework level produced the method-of-approach philosophy discussed in Chapter III. Here is discussed the potential richness of having direct man-to-computer communication become a dialogue. The automation of the symbol manipulation associated with the minute-by-minute mental processes seems to offer a logical next step in the evolution of man's intellectual power. The approach that seems logical is to redesign the capability hierarchies from the bottom up, using a bootstrapping technique. Since the object is to increase human

intellectual effectiveness, one practical test of any redesign is to use it in problem solving. Thus, the basic-research attention has primarily been focused on augmenting those human capabilities that are needed in augmentation research. This approach incorporates a positive feedback, and at the same time, permits the earliest possible attainment of the rich and significant gains foreseen.

One example of the bootstrapping technique is the design and implementation of an off-line, computer-aid system for composing original text. This system, which is described in Chapter IV as the Z-code system, was used for much of the original composition of this report. The techniques were not smooth: the system would occasionally backfire from unwitting violation of a subtle point in convention; many extra hours went into composition of some parts because the system was not compatible with the thought processes of the author; and considerable constriction was experienced. However, the feeling of being tied into a system made a surprising difference in the work itself, revealing new kinds of inadequacies, and new kinds of possibilities.

It must be understood that many of the details of establishing a computer-based laboratory and the basic associated hard- and software needed for the first stages of bootstrapping with computer aid are uninspiring and not necessarily technically relevant. New ventures require the coordination of new people and new approaches toward new goals. The inherent slowness of this process often seems discouraging, just as it is often discouraging to attempt to compose a report with a primitive computer-aided system. However, progress is being made.

II THE MAN-COMPUTER COMMUNICATION CHANNEL

This section shows a specific application of our philosophy to actual research planning, in order to provide some orientation for the generalized discussion that will follow in Section III.

Our research facility is flexible, and is only for experimentation; it is used to implement the trial subsystems that we design so that we can actually employ them in the total process of designing, writing, debugging, documenting, and modifying our own computer programs. To help in launching many levels of concurrent activity, we are implementing first a very crude total system. This will provide an actual working experimental environment to help orient participants by simulating the research activity at the different levels of the total system.

From our experience of designing a first-pass, crude, computer-aided system, we find that little of the work reported on in the literature on man-computer communication has any direct application to this kind of user system. Because of this lack, the coordinated, whole-system approach also demands smaller units of research to fill the specific needs of the system. An example of this is the area of man-to-computer communication.

A. MAN-TO-COMPUTER COMMUNICATION BACKGROUND

1. Computer-Aided Human Communications

First, it proves useful to think of the entire communication subsystem as consisting of subsubsystems involving human transmission and human reception of information. It appears feasible to introduce direct computer aid into these subsubsystems, according to the general possibilities sketched in Figs. 1 and 2.

We assume that the larger activity, the computer-aided communication subsystem, will accommodate the needs for transducing to and from the signal forms required in other parts of the system and that Processes D and E deliver and accept information coded in some

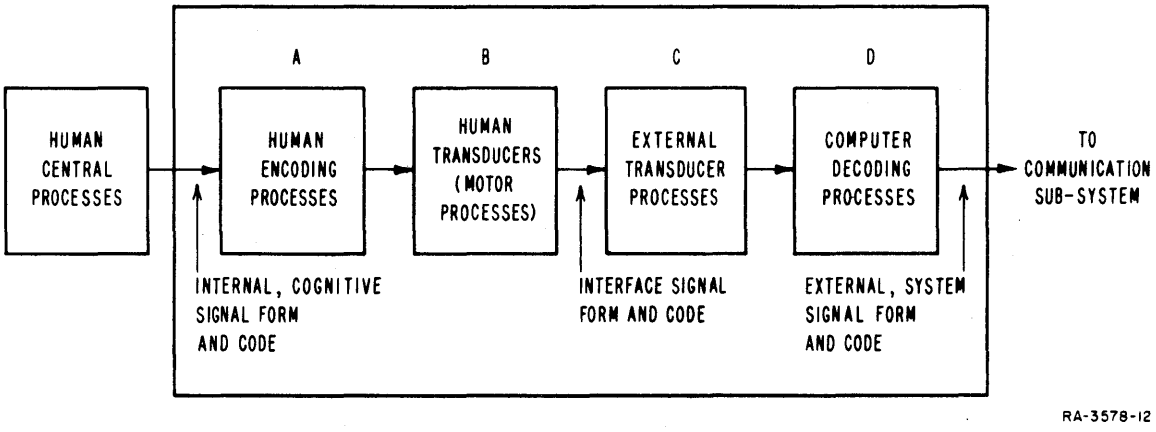

FIG. 1 COMPUTER-AIDED HUMAN-TRANSMISSION SUBSUBSYSTEM

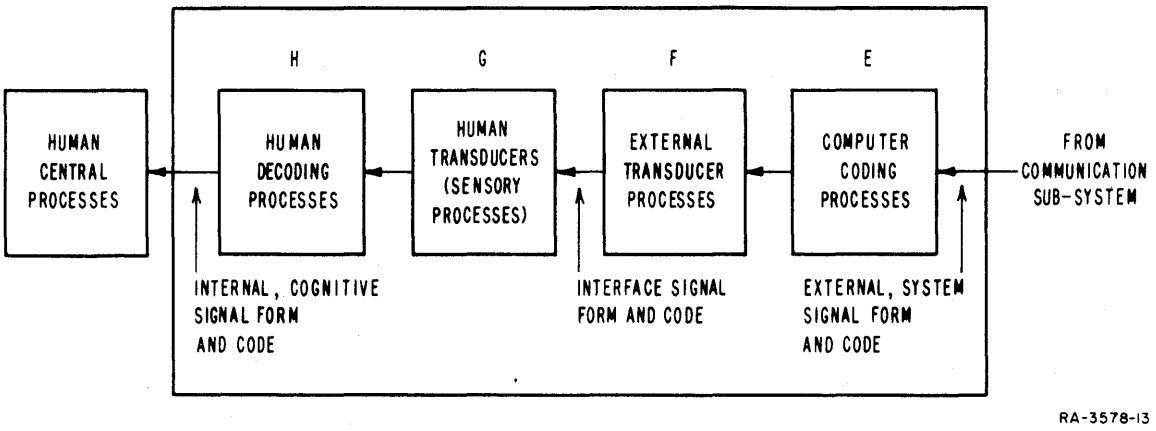

FIG. 2 COMPUTER-AIDED HUMAN-RECEPTION SUBSUBSYSTEM

form directly compatible with general information-manipulation processes within a computer.

The basic aid given the subsubsystems by the external transducers and the computer processes is in providing a match between the signal forms and the codes used in the rest of the system, and the signal forms and codes at the interface that can best be handled by the human capabilities represented by Boxes A, B, G, and H.

When it comes to organizing the human-transmission and human-reception subsubsystems within the human-communication subsystem, we find that (as in Sect. III), we can utilize computer aid in integrating the capabilities of computer-aided lower-level systems into a higher-level system. Figure 3 shows a representation of this that appears to offer completely general possibilities for a human-communication subsystem. The basic aid given the subsystem by computer processing is to provide versatile feedback through the two open-ended subsubsystem channels--to enable more effective use of each channel by means of cooperative use of the other. A simple example would be displaying a representation of a message as the human transmits it.

2. The Open-Ended Subsubsystems

To begin research activity in the human-transmission and human-reception subsubsystems, we look first at the chief design factors: the information characteristics of the messages, the signal forms and the information encoding at the interface, and the computer decoding process. These must end up being compatible with the system design and with human ability to learn and to perform. The message information characteristics depend upon the nature of the larger system; we chose to consider English text transmission, to permit us to go ahead with meaningful and generally applicable activity.

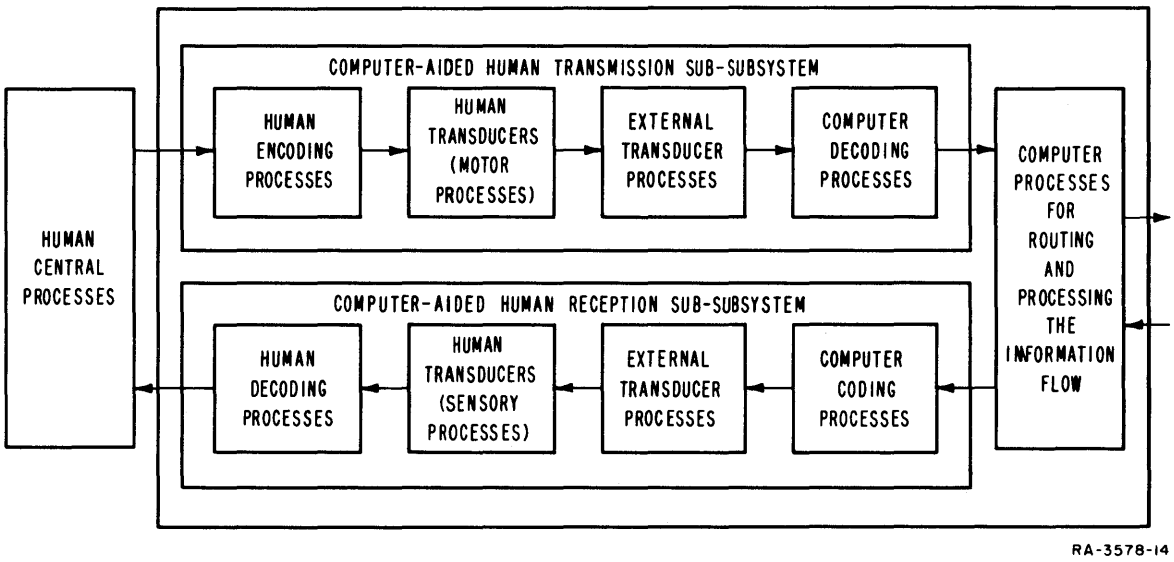

FIG. 3 COMPUTER-AIDED HUMAN-COMMUNICATION SUBSYSTEM

The range of possible body actions from which transducers can obtain computer-sensible signals is very large, as is the range of possible ways to encode information by combinations and sequences of these actions. Similarly, there are large ranges for the types of computer-controlled transducer stimuli that can be used as input signals through human sensory mechanisms, and for the ways in which combinations and sequences of these signals can be used to encode information.

For our research in the man-to-computer communications area, we plan to take off from what we know now about existing techniques and capabilities, and push in a direction that offers both to teach us practical things about human capability and to provide application experience with new communication subsystem features that seem valuable.

B. PRESENTLY KNOWN TECHNIQUES

1. Speech

This we think of as "most natural," and our average transmission rate is quite good--apparently about 180 words per minute. By pushing a bit, I can speak 240 understandable words per minute. The signal forms and encoding techniques are actually very complex--a great many different muscles coordinating in complex combinations and sequences with a very sophisticated, highly redundant encoding. We have no way of knowing how long it will be before a system-applicable speech-recognition machine will be able to decode full English transmission.

2. Stenotypy

Stenotypy is a keyboard-shorthand system, which apparently gives the fastest known transmission of natural language (up to 300 words per minute). The signal actions involve 23 keys with two-handed strokes on the keyboard. At each stroke, different combinations of the keys are depressed, and sometimes one finger may depress two keys simultaneously. Not all of the 2^{23} -1 key-depression combinations can be actuated with ten fingers, but there are over 3,000,000 unique combinations of 10 out of 23 keys, which is still only a subset of those than can be actuated. Information is encoded through a phonetic representation of words. Each two-handed stroke prints some pattern of phonetic vowel and consonant characters across a 23-space line, and each key corresponds to a fixed character imprintable at a fixed location. Three zones across the line provide for initial consonant, middle vowel, and final consonant. Some words require but one stroke, others may take two or more. A relatively tiny subset of the possible signal actions are thus utilized in the code vocabulary, but even so it requires an estimated (by Stenotype school) 52 weeks with a total of 720 hours to train a Stenotypist

to a rate of 125 words per minute, and another 800 hours to raise his speed to 175 words per minute. Because of the phonetic encoding, there are many of the same ambiguities to resolve in decoding as are found with voice. IBM developed a computer-translation technique[3] but although the Stenotype code had to be modified to remove the most troublesome ambiguities, the IBM language translation computer with ten million bits of storage (considered "inadequate") must do the translation, which even then is not perfect.

3. Standard Typewriter

This is the most widespread transmission technique for coupling nonnumeric communication to a computer. Rates of 60 words per minute represent a good average. Training is relatively easy. The signal actions involve about 45 keys, of which one is struck at a time. Good technique assigns each key to a specific finger, and both hands must be held to the keyboard for normal text transmission, and must cooperate synchronously but not in uniform alternation. The encoding is a very straightforward one character per key, with a shift in significance (or character assignment) often available between several cases as specified by the operator through special case-shift keys. Coupling to a computer and conversion to computer-sensible code is straightforward.

4. Telegraph Key

The telegraph key is familiar, but inadequate, with a maximum speed of 25 words per minute. The signal actions involve opening and closing a single contact by means of a spring-return lever. The signals given are one of two lengths of contact-closed period or one of two lengths of contact-open period. Serial sequences of these are encoded to represent characters. More frequent characters are given shorter codes. The code itself is relatively straightforward to decode by computer, but it proves to be very hard for a machine to

---

[3] Galli, E.J., "The Stenowriter--A System for the Lexical Processing of Stenotypy," pp. 187-199, IRE Transactions on Electronic Computers (April 1962)

determine accurately the dot-dash sequence of a human transmission. Besides this difficulty and the slow transmission rate, this technique seems relatively difficult to attain; --it seems to take longer to become a proficient International-Morse code operator than a proficient typist.

5. Handwriting

I seem to be able to write legibly and comfortably enough to do steady composing at a rate of about 30 words per minute. The signal form and the encoding are complex, although not so much so as for speech. Handwriting offers very flexible formatting--insertions and cross-outs are made quite easily.

6. Relative Suitability for Our Needs

Voice and handwriting offer the advantage of being existing skills, but the complex signal forms and coding involved make them poor candidates in a serious search for basically better interface signal forms and codes.

The speed of stenotypy is desirable, but the decoding complexity is a disadvantage. However, the means by which stenotypy provides its range of signal actions is a separate feature that is worth considering. This depression of two or more keys per hand stroke is often termed "chording."

The telegraph key is undesirable because of its limited signal actions and the difficulty in getting more speed even with sophisticated coding. The notion of one-handed transmission, however, is appealing, especially in situations where the human would like to do other tasks concurrent with transmitting.

From the evaluation of existing techniques, a possible new technique can be envisioned. This will be discussed next.

C. THE CHORD-HANDSET APPROACH

The chording approach, using relatively few finger-operated keys to provide many possible signal forms (hand-stroke chords),

seems a very logical first step toward exploring new possibilities at the man-system communication level:

(1) It involves a relatively minor excursion from familiar techniques

(2) It offers interesting system-capability possibilities

(3) It promises to orient us rather nicely to the needs for basic knowledge of human capabilities and for development at other levels of our system.

As a start in this direction, I have done some experimenting with using the 31 unique five-finger chords for transmitting English text. I have devised and learned a five-key code capable of duplicating the upper and lower case typewriter character repertoire and can transmit at a rate of almost 35 words per minute.* Most of my practice has come by drumming my fingers on any handy surface--copying roadsigns as I drive or trying to copy people I happen to be listening to during the day. We have made some five-key handsets, connected them to our CDC 160A computer and programmed it to sample the keys, determine when a signal stroke has occurred, decode this, and type out on the on-line typewriter.

One trial subject, our transcription clerk, learned to transmit at about 20 words per minute after about 25 hours of unsupervised practice. We lately have been using the partial learning of this skill as an experimental task in another project, and find that clerical-type subjects learn very readily to encode alphabetic characters into five-finger chord strokes on our five-key handsets.

A very interesting feature has evolved in this exercise. I find that the learning transference is quite good to my left hand. In fact, with relatively little additional practice, I can transmit left-handed at about 25 words per minute, and have no doubt that the left hand will soon do as well as the right. This system feature of being able to transmit with either hand introduces a new versatility.

*See Appendix B for description of the code.

Beyond this, I have learned without too much effort to transmit with both hands simultaneously, alternating synchronously so that each hand transmits every other character. It gets to be rather like picking up walnuts with both hands, where the "central controller" in one's mind selects successive nuts to pick up, and some automatic internal mechanism synchronously alternates the hands without particular conscious effort. I haven't practiced this even as much as with the left hand alone, yet my two-handed rate is faster (about 29 words per minute). It would be interesting to explore far enough to determine the speed potential of the two-handed mode relative to that of one hand only.

D. DIRECTIONS OF POSSIBLE REVERBERATION AFTER INTRODUCING THE CHORD HANDSET

Experimentation in using the chord handset in connection with activity on our on-line system could produce reverberations (See Section III) at various levels within the system that might include the following:

Development of a ten-key handset on which each finger of one hand can strike one or both of two keys. This would provide 1023 unique chords to use for encoding not only the individual letters but also high frequency syllables, word endings, etc. Using such a handset in daily work ought to provide information on how large a vocabulary of unique codes can be adequately handled, and whether there is some optimum rate at which a user can transmit information.

Development of a mobile handset (perhaps in the form of a glove) which could be used away from the console and concurrently with other tasks. The transmission could be off-line if some storage medium (such as paper tape) were used.

Development of some new reception device using the binary principle of the chord handset to permit responses from the computer to be displayed or transmitted with less translation equipment.

The last of these reverberations would have considerable avalanching effect. Because it is very easy to display binary patterns (visual, audio, or tactile) to a human, the potential ability of a human to receive and decode binary patterns with relative ease is worth serious consideration from the system point of view. CRT displays would not need such expensive character-generators, and simple shift-register, indicator-light packages could provide fast and flexible displays. Very simple binary-pattern printers can give hard-copy output. Or an audio channel could be used in parallel with a visual channel, or by itself to allow more physical mobility unhampered by fixed-position visual display, or as backup in case the visual channel fails. Tactile input offers much the same possibility as the audio.

The signal forms used for these human-input channels can be directly compatible with those used for the human-output channels, and the coding could be identical. Computer operators and punched-paper-tape users often learn to read binary codes quite handily, and psychologists have tested human discrimination capability on binary patterns versus one-out-of-N patterns. As far as I know, nobody has developed coordinated techniques for binary communication within an operational environment and had people use these techniques for their working communication for an extended period.

A further reverberation possibility is practical binary audio computer-to-man communication, which in turn might present possibilities for very cheap on-line work stations and for working aid from a computer under unusual operational conditions. A need would then exist for research into how well a human could work with a computer if he were limited to audio communication from the computer, assuming a rate near that of speech. Full use of feedback and control processes would be important.

III THE USER-SYSTEM, SERVICE-SYSTEM DICHOTOMY

The following considerations are relevant to the general area of man-computer-systems research and development.

In the earlier report for this project[1] there was defined a system that was held to be of basic importance to human intellectual effectiveness. This system was composed of the human together with the language, artifacts, and methodology that he had been trained to use. The language, artifacts, methodology, and training are redesignable in a total-system approach toward increasing the human's intellectual effectiveness.

For some very natural reasons, the area of this "total system" that has received by far the greatest attention and development in the man-computer-systems-research community is the one which would be labeled above as artifacts--"physical objects designed to provide for human comfort, the manipulation of things or materials, and the manipulation of symbols," (from Ref. 2, p. 4). And, most of the workers in the community seem strongly oriented towards artifact development.

To try to develop some perspective in this regard, it seems useful to consider that the systems under study in the over-all research community consist each of two subsystems: Service system and User system. A Service system will provide computer service to a User system at some interface. It is often planned (in time-sharing systems) for one physical Service system to serve many User systems simultaneously. This feature is included here, although it is not emphasized. Most of the following discussion assumes a User system composed of a problem-oriented professional dealing mainly with symbols, although it would not seem difficult to accommodate variations into the framework described.

---

[1] See Page 1

An interface between each User system and its Service system will exist. Across this interface passes the two-way interchange of requests for service (both explicit and implicit) and their consequent service results--in what is useful to consider below as request-response "transactions." At what point between the computer main frame and the human brain this interface is defined is rather flexible, but for the discussion below we would put keys, buttons, light pens, and display screens on the Service system side, together with the conventions and system of the command "language" with which service is elicited from the devices. On the User-system side of the interface are the procedures and processes with which service requests are composed.

It was strongly emphasized in the previous writings that a great deal of evolutionary interaction exists between the total-system components. This belief holds here, too, in that developments in the User system and the Service system each interact upon needs and possibilities of the other.

A. USER SYSTEM

The Service-system, User-system dichotomy leaves the human and the following total-system components in the User-system category (Definitions taken from p. 4 of Ref. 2):

1. Language--the way in which the individual classifies the picture of his world into the concepts that his mind uses to model that world, and the symbols that he attaches to those concepts and uses in consciously manipulating the concepts ("thinking").
2. Methodology--the methods, procedures, and strategies with which an indivudual organizes his goal-centered (problem-solving) activity.
3. Training--the conditioning needed by the individual to bring his skills in using language, artifacts, and methodology to the point where they are operationally effective.

The basic concern of the man-computer research community is assumed to lie with the effectiveness of the User systems in pursuing their objectives. The only valid objective of a Service system would seem to be to increase that effectiveness.

B. SERVICE SYSTEM

1. Parameters of Service

The parameters to the service supplied a User system are assumed to include the size of the different storage media, their access times, computer processing speed, reliability, the service delays, the amount of information visually displayed, the speed and flexibility with which the computer can display it, the repertoire of display characters, the power of the command languages, and so on. These parameters will be assumed to form a multidimensional Service space, with subspaces, surfaces, and the like. A given Service system is assumed to present a given User system with a surface, in Service space, that represents the limits in service that can be supplied (the Service-limit boundary). Service is assumed to be provided by means of request-response transactions.

2. Transaction Points

For a Service system to provide the response to a User system in a given transaction, some given degree of service must be available along each of the dimensions of Service space, so that each transaction is assumed to establish a point in Service space. A User system will initiate many transactions, some large and some small, some nested within others, some straining the Service system to the limit and others exercising less than one percent of the potential. It is thus possible to speak of a scattering of transaction points through the Service space. Areas within the Service-limit boundary surface that are void of transaction points can be assumed to indicate overdesign. Transactions whose points fall beyond the boundary surface, but whose value to the User system outweighs the cost of extending the boundary surface, can be assumed to represent underdesign.

C. CATEGORIES OF RESEARCH

There is a directly evident categorization of research activity provided by introducing the User-system, Service-system dichotomy. Within each of these categories, however, we find "system levels" that suggest further categorization. To facilitate discussion, we have isolated three such levels in each system, labeling them "system," "device," and "material" levels, in analogy with a System-level categorization applicable in many engineering systems. Better categorization could well be valuable to a more extended discussion.

1. Service-System Research

New regions of Service space must be made available, within arbitrarily established cost limits, for the experimental development of User systems, and for pilot installations to be used by working User systems. Eventually, new regions of Service space must be made available to working installations within the cost limitations that are imposed by the estimate of the utility that will accrue to a particular User system in daily work.

Within the Service system, system-oriented research includes the whole-system viewpoint. Device-oriented research would include new memories, new displays, new executive systems, etc., pursued with only background consideration for the whole system. Materials-oriented research would include new phenomena that could be harnessed into new components. Besides physical phenomena research, this would include voice-recognition research, artificial-intelligence research, research into new principles of storage allocation, compiling, and list processing, etc.

2. User-System Research

The system-oriented approach to User-system research involves studying the balanced possibilities for harnessing service transactions available in a given region of service space to the over-all symbol-manipulating activity of a given problem-oriented user. Such an approach would be, for instance, to give maximum over-all aid to a

computer programmer, or to a cryptographer, a designer, an applied mathematician, or a manager.

Device-oriented research involves studying how a given region of Service space can be used to improve some capability that forms only one part of a given type of User system. For instance, I would classify Ivan Sutherland's Sketchpad[4] as a device within the symbol-manipulating domain of a design engineer, and Glen Culler's system[5] as such within the domain of an applied mathematician. They are very bright examples of devices, but their respective User systems still involve far more symbol manipulation than these devices provide. The Sutherland-Culler examples are like exciting new arithmetic units built with a new component technique, which have still to be matched by surrounding devices (control, memory, input/output, etc.) that potentially can be much improved by use of the same new technique. (The new arithmetic units can be fitted into existing systems, but a whole system rebuilt for the new component technology will be much more powerful than the old system retrofitted with a new arithmetic unit.)

Materials-oriented research involves studying basic phenomena of User system organization and components. For instance, techniques for structuring and tagging information, a better language for planning, improving the information-transfer rate of a human, and studying the skill limits of humans would be examples of materials-oriented research in the User system area.

---

[4]Sutherland, Ivan E., "SKETCHPAD: A Man-machine Graphical Communication System," AFIPS Proceedings--Spring Joint Computer Conference, Vol. 23, (1963)

[5]Culler, G. J. and Burton D. Fried, "An On-Line Computing Center for Scientific Problems," Thompson-Ramo-Wooldridge, Inc., Canoga Park, Calif., (January 11, 1963), AD 296582

D. APPROACHES TO THE PROBLEM

1. The Importance of "Value to the User"

One key to research for the man-computer community is the identification of the value to the User systems that can be derived from the real-time computer aid (including intercommunication possibilities). If only limited value can be derived from opening up a given region of Service space, then this service must be cheap if it is to be provided. If startlingly significant value is expected to be derived, initial cost is less important. Experimental working systems of significant value will stimulate interest, create a market, evoke new research, and so on. The resulting experience with the problems and possibilities of actual operating Service systems can be expected to provide better orientation than would otherwise be available for research and development in that area.

Any User system will typically mix many Service transactions with many User actions. Denial of any particular kind of Service transaction may disable some large and critical part of this organization of User actions and Service transactions. The worth of a Service transaction to a User cannot be determined until the structure of his organization of actions and transactions is determined, and until system analysis can isolate the changes in User-system capability caused by denying that type of transaction. When this has been done, a value can be established for the capability. A particular service transaction is justified only if it has enough value in boosting this effectiveness to offset the cost of the transaction. The cost involves both a prorated utility-available cost and a particular-utilization cost, and fairly straightforward accounting would seem to take care of determining this within a given Service system.

But, for example how does one determine the value of having fast response from the computer, and of having CRT-like displays where the display can change quickly and flexibly? Considering this Service facility solely from the point of view of being able

to compose and modify displayed information--to erase that word, move this sentence up and have the rest of the text rearrange itself accordingly, begin a new paragraph there, insert the following word, change that symbol to logical AND--makes it extremely hard to place a value upon that facility until it has been integrated into a coordinated and practiced way of working. A man could predict that he would have no use for such a facility, but until he appreciated, from experience, what it was like to have the added capability, his evaluation would be premature.

2. Special Orientation Toward User-System Research and Development

There is a strong need for a class of researchers who want to design and experiment with new User systems. This class is different from the class of those who are trying to do useful research in some other discipline and are, as a secondary activity, trying to put together some useful new User-system tools and procedures. A characteristic necessary to the User-system researchers is that they be system-oriented. Device-oriented User-system research will often be furthered by the ingenuity of the other-discipline researcher out to make himself some new tools. But the other-discipline researcher lacks the time, background, and interest to become knowledgeable enough either to develop his ingenious devices or to coordinate them in the design of his whole User system.

Significant User-system research need not await fundamental advances in Service-system development. Many of the service transaction points of tomorrow's exotic User systems--basic to many of the fundamental activities of the system--fall within those regions of Service space already available to us. For instance, composing and editing displayed information with really polished speed and flexibility probably will not depend upon having huge memories or very much computation; total-system-oriented subsystem research could thus be emphasized with little need for prior Service system research.

3. Ideal Research Results

It "would be nice" to be able to plot a family of curves, with types of User systems as a family parameter, against an abscissa of Service-system sophistication and with User-system capability as an ordinate. We cannot do it, but there are a few points on these curves that it seems important to attempt to determine. In particular, it seems important to find the highest User-system capability that can be developed from the most exotic Service system obtainable in the research laboratory. This would be a very useful "calibration" experiment. It seems safe to assume that the capability-sophistication curves would rise monotonically--that from any less exotic Service system there could not be derived as much added capability to the User system. If, for such an experiment, relatively little gain in capability could be attained, concentration upon more basic User system research would be indicated. If a very great payoff in capability were found at this extreme experimental point, then we could see where to cut down on the sophistication of the Service system without reducing capability too much, or where to search for intermediate points that could satisfy the economic needs of potential users, and thus move rather directly toward application.

4. The Reverberation Principle

Generally it is useful to consider a system as being composed of many levels of subsystems and subsubsystems (as done roughly above with "system," "device," and "material" levels). In improving a system, an innovation at one level often leads to reverberating waves of (1) possibilities for other innovations, and (2) needs for other innovations. Waves of possibilities tend to propagate upward with an increasingly broad effect--new gains from an innovation (or possibility) at a lower level provide a new innovation possibility (or perhaps several) at a higher level. Gains from these possible innovations are added to the original gains, to stimulate possibilities at still higher levels, etc. The process is similar for innovation needs: consideration of an innovation possibility at

one level often stimulates a new need for functions at lower levels. To fulfill such a need, one or more new innovation possibilities are considered at the lower levels, and again, each is likely to generate new needs in the levels below it, which join with the original to form a downward avalanche of needs.

Each new innovation arising in either the upward wave of possibilities or the downward avalanche of needs is the potential source of a new wave in the opposite direction; it reverberates.

5. The Home-Level Principle

The researcher who is most likely to make significant innovations at a given level is the researcher who is "at home" at that level.

In general, every level of a User system is supported by sublevels while it also supports higher levels; every level thus participates in the upward and downward waves of innovation possibility and innovation need. But every level has its own collection of specific concepts, techniques, and methods and these collections are likely to be unique. Any given level inherits a unique set of needs from the levels above, together with the relative-importance distribution among them. It also inherits a unique set of possibilities from the lower levels, together with associated constraints and relative-potential values. To make significant innovations at that level requires a coordinated knowledge of these needs and possibilities, extending to the "subliminal" level of knowledge that is referred to as "having a feel for" these things.

A researcher cannot be at home in a given level unless he knows enough about adjacent levels that he can deal realistically with the importance, value, and constraints associated with the possibilities and needs from these adjoining levels. It is obvious that the more levels over which he is at home, and the greater his understanding of the interlocking considerations of need and possibility at a given level, the more likely he is to make really significant innovations.

6. Implications for User-System Research

User-system improvement will require, as implied by the above:

(1) Establishing concurrent activity at many system levels

(2) Starting multilevel research as a total activity as soon as possible

(3) Keeping the best of communication up and down between the different levels

(4) Giving the different levels considerable freedom to explore

(5) Supporting exploration and research, in contrast to waiting for clever ideas to buy.

All of these will foster an environment that maximizes the reverberation of needs and possibilities among researchers at the different levels. It is important that there be researchers "at home" in all levels, but the degree of understanding, the orientation, and the coordinated motivations associated with being at home in a given level take years--even in systems that have achieved a certain stability in evolution and in the mix of disciplines involved. For computer-aided User systems of enough value to deserve concerted research, being at home will take longer; the very patterns of needs and possibilities transmitted from other levels will shift rapidly as researchers in other levels struggle toward growing understanding and capability.

Research at isolated levels, while it would initially take less time to organize, would invite establishment of overly restricted or false sets of possibilities and needs for the work of that level. This leads to innovations of diminished significance, and to generation and distribution of needs and possibilities that may be weak or even misleading.

IV THE Z-CODE SYSTEM

A. INTRODUCTION

The Z-code system evolved from the principle of having project personnel develop and use computer aids in our own work, and is presented here to illustrate the process and utility of "boot-strapping."

Our emphasis has been directed toward on-line aids, but the value of a coordinated off-line system became apparent to us as we began to see what problems face anyone who tries to do useful work with the on-line aids if his only coupling to the computer occurs during the expensive and much-in-demand on-line operating time. This Chapter describes the off-line text-editing system that we have implemented.

I have long had the habit of doing much of my "thinking" work on a typewriter, when the nature of the subject permitted it. It occurred to me that if I used a private "tapewriter" (our term for a paper-tape-punching typewriter) in my office for such thought-development I could then feed the paper tape into the computer at the beginning of my on-line working session, and thus take direct advantage of off-line thinking time. When I work at a typewriter, the ideas develop as the words flow, and when the thinking is hard, the sentences get butchered, with much insertion and deletion (pencilled arrows and interlineation, etc.), until they say what my current though requires. After a page or two of this, the real ideas begin to emerge and the whole thing needs reorganizing.

Given this process, and admittedly it is a personal one, I began to wonder if the computer that was going to be working for me anyway, when I took the material on-line, couldn't respond to control commands embedded in my off-line text to effect some of

the changes that are so much a part of my thought-development-process--not to speak of correcting the errors that always show up.

Accordingly, a portion of our AFOSR support was used to design and implement such a system. I use the system in my daily work (indeed, much of the first draft of this report was written this way), and the transcription operator uses it in the preparation of our inter-group memoranda. The system described suffers from a number of faults and inadequacies. We plan in the near future to modify and expand the system along the lines discussed in Part D.

Our off-line system involves a translator program (the Translator) for our 160A that converts the tapewriter output (paper tape) into a new tape that can be used either to produce hard copy on the Flexowriter, or as input data for on-line work at the CRT-display console. The new tape (Translator output) reflects the modifications in format and content specified by the control commands that were written into the tapewriter copy during its origination.

B. DISCUSSION

1. Basic Design Principle

The system was designed on the following important and basic principle relative to tapewriter operation:

Everything about the eventual output from the Translator must be unambiguously discernible from examination of the original tapewriter printed copy.

This requires that the operator does not manually change platen or carriage (or carrier) position, nor touch the paper-tape-feed mechanism. It also implies that all translation algorithms be based purely upon the identity and position of printing characters as they appear on the origination copy. Nonprinting keyboard actions--i.e., SPACE, CARRETURN, TAB, BACKSPACE--are to be considered only as they may be inferred from the copy. For example, SPACES or TABS immediately preceding a CARRETURN must be ignored by the Translator, since they could not be detected by examining the copy. And it

must not matter whether a TAB or successive SPACES produced a given horizontal separation.

2. Some Definitions

Later discussions will be facilitated if some of the special terms are defined here. The origination copy, as represented by the linear succession of punched codes on the paper tape, is considered as a string of characters. This input character string (as seen by the Translator) will be composed of printing strings alternating with gap strings. A printing string is an unbroken succession of (any) printed charcters, and a gap string is a succession of non-printing characters producing the horizontal and vertical separation between two successive printing strings.

Independent of the above sub-string categories (printing and gap), there is another categorization--there are data strings and control strings. Control strings serve as messages to the Translator about the entire input string; they specify both content and format changes.

3. The ZN, Data-To-Control, Escape Code

Except for the four special control characters described in Part C-1, the translator detects all of the control strings by stopping at every numeric character that immediately follows a letter "Z" (either upper or lower case), and checking to see if what follows is a valid command. Termination of delete-command control strings is implicit (but unambiguous), while the remaining types of control strings are terminated explicitly.

We plan to provide future means for handling data strings that look like control strings (see Part D). So far it has not been a problem; we have not suffered for being unable to use certain ZNXX character strings as throughput data.

4. Conventions for Describing Control Strings

When a sample control string is to be presented as data (e.g., in the discussion below about the control strings) without being removed and interpreted by the Translator, punctuation marks are inserted to render the string invalid as a command so that it will be ignored by the Translator. None of the valid command forms includes any punctuation marks--with this knowledge a reader seems to suffer no added confusion from finding punctuation marks in sample control strings presented for illustration.

In the following discussion, the character pairs n1, n2, n3, ... will frequently be used to represent distinct decimal integers (with no pre-established size limits). Since the letter n is not used in regular command-string forms, any appearance in subsequent command-string examples is assumed to be in association with the following digit--the pair to represent a decimal integer.

C. DESCRIPTION AND EXAMPLES

1. Alphabetic Case and Underline from the Model 33ASR Teletypewriter

Because it is far less expensive than other candidate tapewriters, I selected the Model 33 ASR Teletypewriter for my personal use. That it has only one alphabetic case, and has no tabulator, backspace, or underline make it less than ideal, but these deficiencies can be circumvented. The following conventions were adopted initially. They serve reasonably well for general use, and the planned reworking should improve their effectiveness. Four of the Teletypewriter's sixty-three printing characters (/, +, <, >) are used as special control codes when in specified contexts. Each upper-case word below designates a character. SLASH, PLUS, LESSTHAN, and GREATERTHAN represent the above-mentioned control characters. ALPHA represents any alphabetic character, and NONALPHA represents any other character (either printing or spacing). When one of the four special characters appears between two characters of the specified type, the Translator deals with it as indicated below. Found

in any other context, these special symbols are treated by the Translator as ordinary data characters:

NONALPHA. SLASH ALPHA. -- specifies that the SLASH is to be dropped and the alphabetic character made upper case.

NONALPHA. PLUS ALPHA. -- specifies upper case for this and all other alphabetic characters that follow until interrupted by a gap string or a nonnumeric, nonalphabetic character. The Translator drops the PLUS.

NONALPHA. LESSTHAN ALPHA. -- specifies underlining of this and all the alphabetic characters that follow, up to the first nonalphabetic character. The Translator drops the LESSTHAN.

ALPHA. GREATERTHAN NONALPHA. -- specifies underlining of this and all other nonalphabetic characters that follow, up to the first alphabetic character. The Translator drops the GREATERTHAN.

2. Deletion Commands

If the Translator finds a ZN1L control string, it will begin with the last character of this command (the L), and move backward deleting all characters until it has passed the N1th CARRETURN character. Then it continues deleting all NON-PRINTING characters until it finds either a PRINTING or a CARRETURN character, which it leaves. The character immediately to the right of the L of the control string will now appear in place of the beginning character of the string that has just been deleted.

If the Translator finds a ZN1C control string, it will begin with the last character of the command (the C), and delete backward through the control string and through the N1th character preceding the Z. A carriage-return action is counted as one character (a CARRETURN), but tabulator action (represented by a TAB character)

EXAMPLE ZC-1

+EXAMPLE +ZC-1: /THIS PASSAGE, SHOWN IN BOTH ORIGINATION FORM (/TELETYPE) AND IN POST-/TRANSLATOR FORM (/FLEXOWRITER), ILLUSTRATES THE USE OF <UPPER>-<CASE AND <UNDERLINING +CONTROL +CODES. /MISCELLANEOUS EXAMPLES: +CDC +160A; 160/A: +ANFS//Q32; +ANFS/+Q32. /IN GENERAL, ARBITRARY MIXING OF UPPER- AND LOWER- CASE ALPHABETIC CHARACTERS WITHIN ONE WORD IS IMPOSSIBLE TO SPECIFY IN THIS MANNER. /IT ALSO PROVES IMPOSSIBLE TO SPECIFY BOTH UNDERLINING AND UPPER CASE FOR THE SAME CHARACTERS.

EXAMPLE ZC-1: This passage, shown in both origination form (Teletype) and in post-Translator form (Flexowriter), illustrates the use of upper-case and underlining CONTROL CODES. Miscellaneous examples: CDC 160A; 160A: ANFS/Q32; ANFS/Q32. In general, arbitrary mixing of upper- and lower- case alphabetic characters within one word is impossible to specify in this manner. It also proves impossible to specify both underlining and upper case for the same characters.

is treated just as if it were effected by hitting the appropriate number of spaces--where each space is treated as a separate character (SPACE). The movement backward along the character string is exactly as the backspace movement on a typewriter would be if it were possible to backspace over a CARRETURN. (Remember that any SPACE or TAB characters immediately backward from a CARRETURN are to be ignored.) The character immediately to the right of the C of the control string will now appear in place of the beginning character of the string that has just been deleted.

If immediately forward of a ZN1L, ZN2W, or ZN3C control string, there appears a sequence of either N3L, N4W, or N5C, then these latter characters are assumed by the Translator to be part of the command, with interpretation as described below. Any number of successive INTEGER LWC sequences (where LWC is a character that is either an L, a W, or a C) can be thus chained to form a composite delete command. Any deviation from a pure, alternating mix of integers and letters (L, W, or C) will implicitly signal the end of the control string to the Translator, and execution will begin:

(a) The Translator will associate a single integer number with each of the three letters, and will choose the last integer to be paired with that letter in the control string.

(b) The Translator begins with the last character of the control string and deletes all characters as it moves backward through: (1) the control string, then (2) through the L-specified number of CARRETURNS, plus any SPACE-TAB sequence just backward of the last CARRETURN, then (3) through the W-specified number of Gap Strings, and then (4) through the C-specified number of characters (with a CARRETURN plus any SPACE-TAB combination immediately forward of it treated as a single character, and with a TAB treated as though produced by successive SPACES).

In dealing with deletion commands, the Translator scans the input string backward (from the end to the beginning), stopping to interpret and execute each deletion command before continuing the scan. Thus, later deletion commands can delete all, or parts of, earlier commands. A deletion command can actually be embedded (and executed) within an unfinished earlier command, since the Translator does not know that any particular group of characters belongs to a deletion control string until it scans back to the initial Z and then reverses to examine what follows the Z. What it then sees and interprets as a command following the Z is a character string that may well be the product of prior deletion operations.

Since the Translator must at present perform within the limited storage capacity (8,000 12-bit words) of our 160A, we were forced either to limit the span over which a command could be effective (especially the insertion commands described below) or to make a two-pass Translator involving an intermediate paper-tape output. We chose the former.

It had been determined from other of our system considerations that we would want to designate explicitly the start of new paragraphs, and to number them for later reference. We thus use the sequence CARRE-TURN ASTERISK NUMERIC as a signal to the Translator of a new-paragraph beginning. Our present convention is to restrict the range of deletion and insertion commands back to and including the most recent new-paragraph ASTERISK. The Translator processes one paragraph at a time, with a 2000-character limit to paragraph size.

3. Insertion Commands

A Z.1I marks the start of an insertion command. This is followed by a parameter string (whose general form is n1Ln2Wn3C) that specifies insertion-point location, then by the insertion string (the character string to be inserted, which is enclosed in parentheses), and then by a Z.2I, which specifies explicitly the termination of the command.

EXAMPLE ZC-2

*2.1 /HERE IS A SAMPLEDZ1C OF SIMPLE CHARACTER DELETION.

*2.2 /SIMPLE ZORDZ1W WORD DELETION.

*2.3 /SIMPLE LINE DELETION.Z1L

*2.4 /MORE COMPL--OH OH, /I WANT TO PUT SOMETHING ELSE AFTER THE PREVIOUS ENTRY--Z2L

*2.3 /IF BOTH LINE-DELETION EXAMPLES WORKED, ONLY THIS 2.3 LINE WILL SURVIVE.

*2.4 /NOTICE THAT WHEN DELETIONS ARE MADE, THE /TRANSLATORZ6W THE /TRANSLATOR FILLS OUT DELTZ1W DELETION GAPS TO PRODUCE FULL-LENGTH LINES IN THE FINAL TEXT. /MORE LATER ABOUT HOW IT KNOWS WHEN A GIVEN POINT IN THE TEXT IS <SUPPOSED TO HEAD A NEW LINE IN THE OUTPUT COPY.

*2.5 /A MORE COMPLEX EXAMPLE OFX CHARACTER-WORD DELTINGZ2W1C WORD-CHARACTER DELETION.

*2.6 /THE WORD-DELETE COMMAND ACTS THROUGH COMPLETE /GAP /STRINGS--MULTIPLE +SPACES, +TABS, AND/OR +CARRETURNS.XX XXXX XXXX Z3W2C

*2.7 /THE CHARACTER-DELETE COMMAND TREATS +CARRETURN X Z2C AS A SINGLE CHARACTER.

*2.8 /ORDER ISN'T SIGNIFICANT IN THE SEQUENCE OF PARAMETERS IN THE DELETION COMMAND.XXX XXXX XXXX XXXX XZ3C4W

*2.9 /ONE CAN REMOVE OR CHANGE A PRIOR DELETE COMMAND WITH SUCCEEDING DELETE COMMANDS.X XX XXX XXXXZ3WYY Z1W3CCZ ZZZZZZ1C ZZZZ1W Z1W3C ZZ Z

*2.10 /ONE CAN MODIFY THE PARAMETER SPECIFICATION AS IT IS BEING COMPOSED MERELY BY ADDING A NEW SPEC TO SUPERCEDE THE PRIOR ONE.X XXZ2W1C1W Y YY YYY YYYY Z3L3W2COL1C

*2.11 /IF ONE FINDS THAT A +CARRETURN MUST BREAK A WORD IN AN UNFORT Z1WUNATE PLACE, HE CAN CLOSE THE GAP WITH A Z.1C, Z.1W, OR Z.1L.

*2.12 /IF ONE MISSES HIS END OF LINE BELL AND PILES UP CHARACTERS UNREA Z2W UNREADABLY, HE CAN DELETE THE OVERPRINTED CHARACTERS TO BE SURE THAT HIS COPYZ1W READABLE COPY INDICATES WHAT THE /TRANSLATOR WILL DO. /IF HE KNOWS THAT N1 SPACING GAPS OCCURRED BETWEEN THE PRINTING CHARACTERS IN THE PILEUP, THEN HIS NEW LINE SHOULD BEGIN WITH /ZN2W, WHERE N2 IS ONE GREATER THAN N1. /BUT IF HE IS NOT SURE OF HOW MANY /SPACING /GAPS WERE INCLUDED, HE SHOULD START THE NEXT LINE WITH A /Z.2L, TO DELETE THE WHOLE PRECEDING LINE, AND THEN RETYPE THAT LINE FROM THE BEGINNING.

EXAMPLE ZC-2

*2.1 Here is a sample of simple character deletion.

*2.2 Simple word deletion.

*2.3 If both line-deletion examples worked, only this 2.3 line will survive.

*2.4 Notice that the Translator fills out deletion gaps to produce full-length lines in the final text. More later about how it knows when a given point in the text is supposed to head a new line in the output copy.

*2.5 A more complex example of word-character deletion.

*2.6 The word-delete command acts through complete Gap Strings-- multiple SPACES, TABS, and/or CARRETURNS.

*2.7 The character-delete command treats CARRETURN as a single character.

*2.8 Order isn't significant in the sequence of parameters in the deletion command.

*2.9 One can remove or change a prior delete command with succeeding delete commands.x xx xxx xz zzzz zzz zz z

*2.11 If one finds that a CARRETURN must break a word in an unfortunate place, he can close the gap with a z.1c, z.1w, or z.11.

*2.12 If one misses his end-of-line bell and piles up characters unreadably, he can delete the overprinted characters to be sure that his readable copy indicates what the Translator will do. If he knows that n1 spacing gaps occurred between the printing characters in the pileup, then his new line should begin with Z.n2w, where n2 is one greater than n1. But if he is not sure of how many Spacing Gaps were included, he should start the next line with a Z.21, to delete the whole preceding line, and then retype that line from the beginning.

The parameter string interpretation differs for an insertion command from that for a similar-appearing string in a delete command. (In our next system we would make them similar, as discussed in Part D.) The insertion point is referenced by counting lines, words and characters forward from the asterisk that marks the beginning of the current paragraph. The asterisk is the first character of the first word of the first line. A "word" here means a printing string as defined in Part B-2, any unbroken sequence of printing characters bounded at both ends by a Gap String. The parameter string n1Ln2Wn3C is generally interpreted as specifying the insertion point to be immediately after the n3th character of the n2th word of the n1th line.

As with deletion commands, the parameters can be given in any order, and for multiple occurrence of character, word, or line terms, it is the last term which is used by the Translator.

The first and last line, word, or character are quite often the desired target for an insertion point. Straightforward interpretation of the parameter string (as above) makes it easy to specify the first item of a given kind, but specifying the last item would require careful counting. To make last-item specification also easy, the following complication has been introduced into the parameter-string interpretation.

If any term is unspecified in the parameter string, the Translator will interpret this as designating the last item of its class. For instance: omitting the line specification designates the last line (i.e., the one in which Z.1L appears); omitting the word specification designates the last word of the designated line; omitting the character specification designates the last character of the designated word. Zero-line and zero-word specifications are equivalent to omitting the line and word specification, but zero-character is interpreteddifferently from omitting the character. A zero-character specification will put the insertion point in

front of the first character of the designated word, while omitting the character specification puts the insertion after the last character.

The insertion string can contain both data and control strings (except for insert-command strings, which cannot be handled within insertion strings). The control strings will be interpreted as if they occur in their inserted location. The Translator moves backward through the input string, interpreting and executing control strings as it finds them. An insertion command thus causes its insertion string to be moved at the time the Translator reaches the Z.2I, but any control strings in the insertion string wait for interpretation until the Translator later reaches the insertion point.

If successive insertion commands are to be given, the Z.1I of the second command can be omitted if the associated parameter string follows immediately after the Z.2I termination for the first command. There must be a gap string after the final Z.2I termination.

CARRETURN appearing in an insertion command has an effect that depends upon where it appears. Within the insertion string, a CARRETURN becomes but one of a string of characters being inserted at the insertion point. Elsewhere in the command, a CARRETURN is ignored if it follows an alphabetic character (but will invalidate the command if it follows a nonalphabetic character--an unnecessary inconvenience to be remedied in the future).

Delete commands, appearing elsewhere than in the insertion string, will invalidate the insertion command (another unnecessary restriction). Correcting a mistake in an insertion command is not generally as easy as elsewhere, and may involve aborting the command with a delete command and starting over.

4. Line and Left-Margin Format Control

Using arbitrary deletion and insertion capability would often result in unacceptably wide variation in length of the output lines. Insertions into a line, or the deletion of one or more CARRETURNS, can often produce a very long output line. Or, most of an input

EXAMPLE ZC-3

```
*1  /A SIMPLE INSERTION FIRST.Z1I4W( EXAMPLE)Z2I
*1  A simple insertion example first.

*2  /DELETE AS AN EXAMPLE A GIVEN PRIOR WORD.Z1I7W(Z1W)Z2I
*2  Delete as an example a prior word.

*3  /REPLACE ONE ITEM WITH ANOTHER.Z1I4W(Z1W WORD)Z2I
*3  Replace one word with another.

*4  /INSERT WITHN A WORD.Z1I3W4C(I)Z2I
*4  Insert within a word.

*5  /SUCCESSVE INSERRTIONS AND CORRECTIONS.
Z1I1L2W(Z2CIVE)Z2I1L3W5C(Z1C)Z2I
*5  Successive insertions and corrections.

*6  INSERTING CONTROL CHARACTERS.Z1I2WOC(/)Z2I3WOC(+)Z2I
*6  Inserting CONTROL characters.

*7  /TRY INSERTING INTO A PRIOR Z1I3W( NEW)Z2I INSERTION
STRING.Z1I1L7W7C(SOMETHING )Z2I
*7  Try inserting something new into a prior insertion string.

*8  /RESPECIFY PARAMETER TERMS IN INSERTION COMMAND.
Z1I2L3W5W1L( AN)Z2I
*8  Respecify parameter terms in an insertion command.

*9  /CORRECTIONS DURING WRITING OF INSERTION STRING.
Z1I1L2W1C(Z1W  /MAKE SME MISZ2W SOME VARIOUSZ1W SAMPLE )Z2I
*9  Make some sample corrections during writing of insertion string.

*10  /IF INSERTION-COMMAND ERROR IS BEFORE THE OPENING PAREN OR AFTER
THE CLOSE PAREN OF THE /INSERTION /STRING (AND ISN'T SUCH AS TO BE
CORRECTED AS IN *8), USUALLY HAVE TO ABORT THE COMMAND AND START OVER.
Z1I1LWZ6CZ1I1L2W( AN)Z2I
*10  If an insertion-command error is before the opening paren or after the
close paren of the Insertion String (and isn't such as to be corrected as in
*8), usually have to abort the command and start over.
```

line may be deleted, to produce a very short output line. Similar troubles can occur with horizontal spacing in tabular data. The following conventions and Translator features have been adopted to alleviate these problems.

The CARRETURN ASTERISK NUMBER sequence that heads a paragraph anchors that asterisk at the start of a line. Within the Translator, in the normal case, the other CARRETURNS are replaced by SPACES, and some other SPACES are replaced by CARRETURNS to produce new lines of uniform length. The Translator can be set at run time for the number of characters to be in the maximum-length output line. The output lines then are terminated at the end of the last full "word" (printing string) that will fit within this line-length specification.

Another provision was made, for left-margin control, that provides a first step toward tabular control, and that also has a feature helpful with new-line control. A control string ZnlT (which is removed by the Translator) establishes the "normal" left margin as being inset by nl TABS. In succeeding text, until a new margin-control command is given, any CARRETURN followed by other than nl TABS is assumed by the Translator to be a specified new-line point. When the CARRETURN is followed by nl TABS, the Translator considers this gap string (CARRETURN nl-TABS) as equivalent to a SPACE. Between specified new-line points, the Translator then replaces SPACES (or equivalent) with CARRETURN nl-TABS wherever needed to produce full lines.

This feature, even when no margin inset is desired, proves to be very useful for inserting specified new-line points without having to use the CARRETURN ASTERISK NUMERIC sequence (which often is undesirable since it establishes a barrier to modification commands, and also may not be desired in the text structure). No TAB-specification control code (i.e., a code which directs the Translator to install TABS) has yet been established for teletype use--but with what always amounts to zero-TAB INPUT, one can still take advantage of the new-line-specification feature of the margin-control codes.

A new-line point is inferred by the Translator whenever more than one CARRETURN is given in succession.

It should be noted that the margin-control codes are interpreted and executed by the Translator, after all deletions and insertions are executed, on a second pass over the character string, this time moving forward from the beginning toward the end of the string. Insertion and deletion commands occurring later in the text can thus be used to modify earlier specification of margin control.

There proves to be a fairly simple way to produce sections of inset text from Teletype input. A Z.1W (or Z.1L or Z.1C) at the start of a new line will delete the preceding CARRETURN. If this is done on successive lines after inputting a Zn1T with n1 greater than zero, the resulting input is treated by the Translator as one long line to be cut into a succession of shortened lines, each inset by n1 TABS. The first line, however must be inset by SPACES to the desired starting point.

D. CRITICISMS AND SUGGESTIONS FOR IMPROVEMENT

Since we plan to keep improving the system, we include with the identification of each inadequacy, notions for remedying it. Two general types of inadequacy exist. First, it is impossible or inefficient to do some of the necessary tasks that are within the reach of the initial system, and second, the reach is limited.

Using the Z NUMERIC sequence to signal the start of control strings leaves the problem of how to designate output text that might have valid command strings in it (strings one doesn't want the Translator to execute). One solution is to use Z1Zx (where x is any printing character) to specify that, moving backward, no control codes are to be obeyed until just forward of the next occurrence of the character "x." Whether or not x is to be removed along with Z1Zx is still open. Insertion could be allowed into this "dead" region, and a delete command initiated after its end could delete into or through the region.

Another solution is to use one printing character exclusively for signalling a control-code occurrence. For the Model 33 ASR Teletypewriter,

EXAMPLE ZC-4

```
*4  /ONE CAN FORCE A NEW LINE WITH:  Z1T
 4.1  /A +CARRETURN +ASTERISK +NUMERIC SEQUENCE, OR
 4.2  /A MISMATCH BETWEEN THE CURRENT +ZN1T SETTING AND
Z1L THE NUMBER OF +TABS THAT FOLLOW THE +CARRETURN.  (/THIS
Z1W LATTER CHARACTERISTIC, TOGETHER WITH THE PRACTICE OF
Z1C DELETING +CARRETURN 0-+TAB SEQUENCES WITH DELETE CODES AT
Z1L THE BEGINNING OF EACH LINE, ALLOWS LEFT-MARGIN INSET OF A
Z1W BODY OF TEXT.) Z1I2L6W( WHICH DESIGNATES THE START OF
Z1W A NEW PARAGRAPH,)Z2I
```

*4 One can force a new line with:

4.1 A CARRE"URN AS"ERISK NUMERIC sequence, which designates the start of a new paragraph, or

4.2 A mismatch between the current ZN1" setting and the number of "ABS that follow the CARRE"URN. ("his latter characteristic, together with the practice of deleting CARRE"URN 0-"AB sequences with delete codes at the beginning of each line, allows left-margin inset of a body of text.)

it is tempting to use BACKSLASH for this, using it in place of the Z in every control command. Initially it seemed advantageous to use symbols found on all of our printing and displaying devices, so that a common set of control codes could be used among them all. Experience shows that if the format and significance of the rest of the control command are constant, different control-flag symbols would be easily accommodated by users; it now seems better to use whatever flag symbol is most convenient to a given device.

Another problem concerns the underline and case control codes for the Teletypewriter. The current prefix symbols serve very nicely for straightforward use, but allow no arbitrary specification of case or underline. One solution is to have case-shift Z-code control strings--e.g., Z2K to specify that all alphabetics to follow are to be upper case, and Z1K to specify return to lower case. Similarly, Z2U specifies beginning of an underline, with Z1U specifying termination. The Translator would remove these control strings. The present prefix codes, or some similar version of them, would be worth keeping because of their convenience for the situations where they do suffice.

The absence of a TAB key in the Model 33 Teletypewriter is a limitation that proves especially bothersome when writing computer code. For an initial solution, we plan to have the Translator interpret the BACKSLASH character as though it were a TAB. This allows the desired explicit specification from the input keyboard, but having no associated spacing gap appearing in the input copy may prove bothersome in rereading prior text.

Although the deletion facility seems generally quite effective, several kinds of suggestion have arisen. One, allow an abbreviation of the oft-used Z.1W code by a single character, still retaining the Zn1W facility. Our current suggestion is to use the dollar-sign character (DOLLAR). Any occurrence of the two-character sequence, PRINTCHAR DOLLAR, is to be interpreted as a Z.1W with backward deletion to begin with DOLIAR. Multiple occurrence might be allowed

to designate multiple-word deletion. This would be less efficient than ZnlW for more than three words, but command composition could often be done as sort of a tallying action while ticking off the word steps to be deleted, for an actual saving in time.

There would also be benefit from a single-character equivalent of Z.lC, with multiple-occurrence repetitive interpretation. Perhaps either or both PERCENT PRINTCHAR or NONNUMERIC PERCENT sequences could signal the delete-character code (with backward deletion beginning with PERCENT). This convention would ignore normal usage of PERCENT--i.e., NUMERIC PERCENT NONPRINTCHAR. The convention for interpretation of the sequence DOLLAR PERCENT might be established as either (1) delete DOLLAR, or (2) after deleting DOLLAR and its preceding word, delete percent and the now-preceding character. The latter may be the more useful convention--we could always use ZnlC to delete unwanted DOLLARS or PERCENTS.

For both delete and insert commands, changes to Parameter String convention and interpretation could make for more efficient and consistent designation. Allow the use of minus-sign characters, so that the n1, n2, ... parameters may be either positive or negative integers. For either type of command, the parameter string will specify the location of a "designated point" relative to a "reference point."

For a parameter string n1Ln2Wn3C, the designated point is found by starting at the reference point, counting to the beginning of the n1th interline spacing gap, from there to the beginning of the n2th inter word spacing gap, and from there through the n3th character. But for each parameter, the direction along the data string depends upon the sign of that parameter: backward for positive, forward for negative.

What reference point to use depends upon the sign of the highest-ranking non-zero parameter--i.e., which direction one would start moving (as interpretation is described in the foregoing paragraph) to find the designated point. If this parameter is positive (the first move is backward), the reference point is just before the leading Z of the command string. If this parameter is negative

(the first move is forward), then the reference point is just before the asterisk that heads the paragraph.

A delete command will remove the command string and all characters between it and the designated point. When all parameters of a delete command are positive, the action will be exactly the same as it would be for the current Z-code System. But a Z.2L-1W, for instance, would delete up to the first word of the line preceding the command, and a Z.-1W would delete up to the first word of the paragraph.

Insertion commands using all-negative parameters would be interpreted exactly the same as the current Z-code System would interpret one using all-positive parameters. In the changed system, one can count lines, words or characters from whichever end of the paragraph, line or word is nearer. For instance: the command, Z.1I3W(XX)Z.2I would insert XX at the end of the third word backward from the command; Z.1I-2L2W(XX)Z.2I would insert at the end of the third-from-last word in the second line of the paragraph; Z.1I-2L2W-1C(XX)Z.2I would insert XX at the beginning of the second-from-last word in the second line of the paragraph; Z.1I2L-3W(XX)Z.2I would insert XX at the end of the third word of the second line above the command.

It also seems that the following addition to parameter-string convention is worthwhile. Beside the n1L, n2W and n3C terms, allow an n4Xab...cX term--where n4 can be either a positive or negative integer. This will be called a search-string term. The sequence of characters ab...c is the search string and X is the delimiter character. We shall allow X to be any nonnumeric, nonalphabetic, printing character other than MINUS, but it must always be preceded by a numeric character. For any such parameter term, the search string ab...c may include any character (printing or nonprinting) except the character used for X in that term. Allowing any of several delimiter characters to be used removes much of the restruction which this latter limitation might otherwise impose.

If X is one of the nonnumeric, nonalphabetic characters from the top rank of keyboard keys, the search-string term will be

interpreted after the line term and before the word term. If X is a nonnumeric, nonalphabetic printing character from any key not in the top rank, then the search-string term will be interpreted after the word term and before the character term.

Interpretation of a search-string term (n4Xab...cX) involves moving backward or forward (for positive or negative n4 respectively) to the n4th occurrence of the character string ab...c, and to come to rest just forward of the c character. Use of search-string terms should save some of the time and tedium of counting*.

For example Z.2Ll"." would delete up to the last period in the second line above the command. Z-2"." would delete up to the second period that appears in the paragraph, no matter what line. A COMMA PERIOD COMMA string could have replaced the QUOTE PERIOD QUOTE string without change of interpretation in either of the above examples, but not so in a Z.2Ll"."1W example, which deletes one more word than do the first examples. A frequent application, for instance, would be such as using Z.1I"en "(Z.1W common)Z.2I for specifying that the word frequent at the start of this sentence be replaced by common. This could also have been specified with parameter strings 1"app"1W or 1","-2W.

With the minus-sign convention within parameter strings, an overlay grid (or equivalent mechanism) fixed onto the Tapewriter could be used to advantage, making always available at a glance the n1-lines-up and n2-characters-right counts to any desired insertion or deletion point, on the recent lines of text, from a reference at the head of the current line. If the vertical scale

---

*This feature is basically borrowed from the COMIT programming language. The nth occurrence and arbitrary delimiter-character (x) features were added here (We have since found the latter to be implemented in the recent on-line typewriter editing system at Project MAC by Saltzer[6]).

[6]Saltzer, J. H., "TYPESET and RUNOFF, Memoranda Editor and Type-Out Commands," CC-244-2, MAC-M-193-2, (January 11, 1965).

started with ONE at the current line, and the horizontal scale started with ONE at the point just before the first character of a line, then an n1L-n2C parameter string would accurately specify an n1, n2 coordinate point on the overlay grid. From my experience, I would judge this to be a very handy innovation--at least with a tapewriter for which the typing carrier moves instead of the platen carriage.

Use of insert commands has been hampered by the fact that it is sometimes hard to repair errors within them. The DOLLAR and PERCENT delete conventions suggested above could improve this satisfactorily, if the rule were that anywhere in an insert command the Translator would interpret and execute these characters (but not Z-codes) before it interpreted and executed the insert command. For instance, Z.1I2L3W(%4W(,)Z.2I would be thus corrected before interpretation to be Z.1I2L3W4W(,)Z.21, and the Translator would use the 4W as the word term instead of the 3W (since only the last-specified term of a given parameter rank is used by the Translator). This rule would prevent usage of DOLLAR and PERCENT characters as a desired part of a search-string parameter term, but initially this constraint would seem preferable over the alternatives of (a) no deletion means within a parameter string or (b) more complex conventions.

It would be useful to be able to specify the right-margin setting with control codes. Perhaps it would suffice to have something like a Zn1R specify that what follows is to have the lines of its translated copy terminated in accordance with an n1-character maximum-length line.

In the second category of inadequate features for the present system, we treat the matter of the "reach" which the Translator can accommodate, back into prior material, to execute its insert, delete and move commands. A reach only back to the head of the current paragraph is a very bothersome limitation.

One level of improvement, to be made as soon as possible, is to extend the reach back to the beginning of the "current" input string--in such a manner that a prior Translator--output tape, or other input tapes, could be cascaded with a new tape and treated as one long input string. With this change there should be added an n1P term to those

acceptable in a parameter string--a term interpreted before any of the others, designating a move from the reference point to the beginning of the nlth interparagraph spacing gap.

A second level of innovation to increase the reach would be to store a record of each Translator output printing in a magnetic-tape (or disk) file. Make the file accessible to the Translator, so that control commands in a new input can specify extraction from any prior printout and integration (with modification) into a new printout. Special control commands could be established to remove any unwanted records.

One could specify collection of many extracted passages, modified as desired, to be integrated with new input material into a new printout. A suitable convention for unique identification of specific records in the file, and of specific reference points in the records, must be established and made easily useable from inspection of the associated hard-copy printout.

With the longer-reach capability, the matter of recycling can properly be considered. (The most frustrating part of my use of this system to date has been that I could get computer help on only the first pass of origination work.) With the extended-reach facility (even at the first level of improvement) one can have computer-aided manipulation over successive drafts as a design record (or memo, report, etc.) evolves into shape.

To accommodate this recycling, the Translator must not remove the format-control codes except on a final-output pass, where standard, clean text is required. It is planned to use a high speed line printer for printing intermediate copy of larger-sized material, and to use control-specification conventions similar to those for the Model 33ASR Teletype to accommodate the one-alphabetic-case and no-underline limitations of the printer.

SPECIAL NOTE: At publication time for this report, an extended-reach system, with recycling capability, is being implemented under SRI internal sponsorship.

E. EVALUATIVE COMMENTS

In forcing myself to use the Z-code System in the generation of a report, I was both hobbled and stimulated. The foregoing change recommendations will go far to alleviate the hobbling, especially if fast recycling can be obtained for several-page sections of text.

I found that for stretches where the words flowed relatively easily, the system was quite helpful, and with later recycling capability I think it will prove to be very helpful over a wide range of writing and rewriting applications. For tougher stretches of composition, the system got in the way. For some of these sections, I went back to more primitive means, a ball-point pen, with lots of scratch-outs, arrows, and marginal scribbling and a skilled typist to translate to clean text. This was much preferable--and I admit this with no sense of failure regarding the utility of the eventual improved Z-code System.

Indeed, viewed as a first in a succession of stages in a bootstrap-evolving, augmentation system, the Z-code System is a success. It revealed and put into perspective in a most satisfying way various needs at lower levels and possibilities at higher levels of my augmentation system. Data structuring (discussed in a previous report[1]) is showing itself to be an important factor in my specific collection of "needs and possibilities." One of the problems in hard-think work is the matter of clarity of presentation of preceding work. Cleaned-up copy via frequent recycling is one help, but brevity in substantive statements, and clear and concise designation of interrelation among statements are also important. These things can be facilitated with better structuring conventions.

SPECIAL NOTE: The new system, being implemented at final proofing time of this report, is designed especially to manipulate text in which relationships among the individual statements are explicitly designated within a crude but formalized "structuring system."

---

[1] See Page 1

It is also very apparent that the "immediate recycle and display" characteristics of on-line text manipulation (at a CRT console) will provide a significant utility in the hard-think type of work. We are developing such on-line facilities, and then plan to keep them to be completely compatible with the off-line system of procedures and conventions as both systems evolve. A person will be able to feed material into the computer, have it processed by the Z-code Translator, inspect, add, modify, etc. from the CRT console, and will be able to output copy that is again available for either or both off-line or on-line recycling.

For some of the time-sharing systems providing remote typewriter consoles such as at MIT (Project MAC), at Systems Development Corporation, at Carnegie Institue of Technology, and at Stanford University, I should think many of the features of our Z-code System (implemented or planned) would be easily implemented and very useful. But also, off-line text-manipulation systems similar to that discussed above could be implemented for use with almost any conventional digital-computer facility. There would seem to me to be considerable potential worth to exploring and developing such possibilities, for use in both clerical and professional activity. The researchers who develop the structuring and procedural conventions best harnessing computer service, and the variety of users integrating these into their way of working would both be getting very good experience and orientation toward the next step--that of on-line text manipulation.

V SUMMARY AND CONCLUSIONS

A. SUMMARY

A specific discussion of man-computer intercommunication needs is developed in Chapter II to illustrate one possible solution. A general ten-element intercommunication model evolves, and a chorded handset is selected for the next step in exploration. Experience with such handsets, and the associated binary codes generates a set of needs and possibilities at this and other User system levels.

In Chapter III are developed some general concepts which are relevant to planning of research in the man-computer area. Immediately introduced is the dichotomy in a whole man-computer system that is represented by the definitions for Service system and User system--with associated concepts of request-response transactions, transaction points in service space, and User-system actions. A chief need in the man-computer research area is exploring the value to the over-all capability of the User system that can be derived from integrating into it particular transactions with the Service system. A system of this kind should be viewed as a many-level hierarchy, and research to improve the system involves "needs" and "possibilities," propagating down and up the hierarchy with "avalanching" and "reverberating" characteristics.

The Z-code System described in Chapter IV is one man's off-line, computer-aided text-editing system--"text" meaning generally anything one might generate at a typewriter keyboard. The user sits at a paper-tape-punching typewriter to produce his text. At any point on his page, as he makes mistakes, changes his mind, or wants to specify some formatting mode, he simply types explicit, visible sets of characters that specify his wishes. His keyboard input is recorded stroke by stroke on punched paper tape, which is subsequently processed by a computer to

locate, interpret, and execute the commands he has thus buried in his text. The computer produces a new paper tape from which a Flexowriter can type the cleaned-up text.

This system was used for the origination of much of the text of this report, and the improved and expanded versions of the system described in Chapter IV-D are planned as a coordinated portion of a larger, continuously evolving working system, which also will include the CRT-console, real-time aids to text manipulation.

B. CONCLUSIONS

Our first primitive system of computer aids will serve us well as the first stage of an evolving progression, and has furnished us much orientation and stimulation. It is not yet competitive with conventional methods as an aid to "hard-think" work, but the recommended improvements are expected to make it so. The modifications and expansions recommended would provide an off-line system with considerable potential to a wide variety of users having access to conventional computation services.

In a more general vein, with the assumption that significant undiscovered and untapped potential awaits us in the man-computer area, it seems very evident that strong, specific focus on User-system research is overdue. The concepts and recommendations of Chapter III are an initial guide to such research, but these would soon be swept into primitive history by the flood of new concepts and developments resulting when attention and energy are focused on this area.

Appendix A

LINKED-STATEMENT DESCRIPTION OF THE Z-CODE TRANSLATOR

Appendix A

LINKED-STATEMENT DESCRIPTION OF THE Z-CODE TRANSLATOR

Assume that the Input String has been produced and is now ready to be processed by the Translator. The paper tape containing the coded Input String is fed into the computer first-generated-end first. The Translator reads this tape, processes the data, and produces new output tape in batches. The following terms need definition: beginning and end of text refer to the first and last produced at origination time. The forward direction is toward the end; the backward direction is toward the beginning.

1. Read paper tape forward, loading into the working store.
   If started at end, HALT--Translation complete.
   If find sequence CARRETURN ASTERISK NUMERIC, go to 2.
   If reach END, go to 2.
2. Start at end of text in working store.
3. Scan backward looking for control commands.
   If reached beginning and found none, go to 5.
   If one is found, go to 4.
4. Execute delete and insert commands.
   4.1 If invalid command, go to 3.
   4.2 If not a Delete Command, go to 4.3.
   - 4.2.1 Determine parameters (n1, n2, n3) of Delete Command, specifying deletion of n1 lines, n2 words, and n3 characters.
   - 4.2.2 Delete the Deletion-Command character string.
   - 4.2.3 If n1 is zero, go to 4.2.5.
   - 4.2.4 Count backward through n1 CARRETURNS, through any SPACES and TABS (in any order) beyond that, and delete everything in this interval.
   - 4.2.5 If n2 is zero, go to 4.2.7.
   - 4.2.6 Count backward through n2 Gap Strings and delete everything in this interval. A Gap String is any unbroken sequence of SPACES, TABS, and

CARRETURNS, in any order and of any length greater than zero characters.

4.2.7 If n3 is zero, go to 3.

4.2.8 Count backward past n3 characters and delete everything in this interval. Each CARRETURN and each SPACE is counted as one character, and a TAB is counted as being executed by a string of SPACES. Go to 3.

4.3 If not an Insert Command, go to 3.

4.3.1 Determine the parameters (n1, n2, n3) specifying the point of insertion--n1 lines, n2 words, and n3 characters.

4.3.2 Determine the designated line: If n1 is unspecified or zero, take the line in which the insert command was started. Otherwise, go backward to the first new-paragraph designator (the sequence CARRETURN ASTERISK NUMERIC), take that CARRETURN as the first and count forward past the n1th CARRETURN. The next character heads the designated line.

4.3.3 Determine the designated word in the designated line: If n2 is unspecified or zero, take the last word of the designated line (i.e., go forward to the next CARRETURN, then backward to the start of the first GAP STRING or to the head of the line, whichever is found first). If n2 is not zero, take the Gap String which includes the CARRETURN heading the designated line as the first, and count forward to the end of the n2th Gap String. Position now is on the last character of the Gap String preceding the designated word.

4.3.4 Determine the designated character of the designated word: If n3 is unspecified, take

the last printing character of the word. Otherwise, take the present position (the last character of the Gap String preceding the word) as zero, and count forward to the n2th character (n2 may be specified as zero, in which case the designated character is the Gap-String character immediately preceding the designated word).

Each SPACE and each CARRETURN will be treated as a character, and a TAB will be treated as if it were executed by successive SPACES. TABS and/or SPACES preceding a CARRETURN are ignored.

4.3.5 Remove the command-specification characters of the insertion control string, and insert the insertion-string part of the Insert Command just forward of the designated character. Go to 3.

5. Start at beginning of text in working store. Set CHCOUNT to zero. COMMENT: This second pass readies the batch for output punching. The description is as for converting from Teletype input to Flexowriter output, where underlining and alphabetic case are converted from prefix specification to the real thing. There are a number of Control States effective during this pass that can be defined for clearer understanding of the process description:

(a) CASE--determines output case for alphabetic characters:
LOW--every alphabetic character to be lower case.
UPALPHANUM--every alphabetic character to be upper case until first character that is neither alphabetic or numeric.

(b) UNDERLINE--determines whether characters are to be underlined:
OFF--no underline.
FORALPHA--every alphabetic to be underlined until reach first nonalphabetic character.

NONALPHA--every nonalphabetic character to be underlined until first alphabetic character is reached.

(c) LEFTMARGIN--determines the number of TABS for "normal" left-margin inset. Variable of the State is: nl-TABS--specifies decimal number of TABS for inset. Nl may be any integer (0,1,2,...).

(d) LINELENGTH--Number of characters to constitute maximum-length line in output text, a parameter set into computer at time the Translator is to be run.

6. Scan forward, detecting and executing remaining control strings.

6.1 Pick up next character in working store. Look at next character.

6.2 No next character--end of working-store text: Go to 1.

6.3 If character is ALPHABETIC, go to 6.8.

6.4 If preceding two characters were ALPHABETIC GREATERTHAN, set UNDERLINE State to NONALPHA. Remove GREATERTHAN.

6.5 If character is not NUMERIC, set CASE State to LOW.

6.6 If UNDERLINE State is NONALPHA, underline the character and go to 7.

6.7 Set UNDERLINE State to OFF. Go to 7.

6.8 If preceding two characters were NONALPHABETIC SLASH, remove SLASH and go to 6.13.

6.9 If preceding two characters were NONALPHABETIC PLUS, set CASE State to UPALPHANUM, remove PLUS and go to 6.13.

6.10 If preceding two characters were NONALPHABETIC LESSTHAN, set UNDERLINE State to FORALPHA, and remove LESSTHAN.

6.11 If this character completed a string Z INTEGER T, set variable of LEFTMARGIN State to the value of the integer string.

6.12 If CASE State is not UPALPHANUM, go to 6.14.

6.13 Make this character upper case.

6.14 If UNDERLINE State is FORALPHA, underline the character, and go to 7.

6.15 Set UNDERLINE State to OFF.

7. Replace unnecessary CARRETURN TABS sequences with SPACE.
   - 7.1 If this character is not a CARRETURN, go to 8.
   - 7.2 If it is, remove any SPACES and/or TABS immediately preceding it.
   - 7.3 If the CARRETURN is followed by exactly the number of TABS specified by the LEFTMARGIN variable, go to 7.5.
   - 7.4 To 9.
   - 7.5 If, in the text forward of this CARRETURN, there are any other TABS bounded ultimately by printing characters, before the next CARRETURN, go to 9.
   - 7.6 Replace the CARRETURN nl-TAB string by a SPACE.

8. Insert CARRETURN Nl-TAB strings where new lines are necessary.
   - 8.1 Increment CHCOUNT by one.
   - 8.2 If CHCOUNT equals RIGHTMARGIN, go to 8.4.
   - 8.3 Add the character to the Output-Line Buffer. Go to 6.
   - 8.4 Scan backward in working store to beginning of closest Gap String. In the Output-Line Buffer, replace this Gap String and the partial Printing String just forward of it by a CARRETURN.

9. Punch out and clear the Output-Line Buffer. Put nl TABS at head of Output-Line Buffer, where nl is the variable of LEFTMARGIN state. Set CHCOUNT to nl times TB, where TB is the number of SPACES between the tab stops on the Flexowriter. Go to 6.

Appendix B

BINARY-KEYSET CODE SYSTEM

Appendix B

BINARY-KEYSET CODE SYSTEM

This describes the current code utilized with a five-key chorded handset. The chart of Table I shows the assignments of meaning given each of the 31 unique chord strokes that are available with five keys. The meaning of each hand-stroke keying configuration (chord) depends upon the current "interpretive case." It is assumed that a computer will interpret these codes, although there may be paper-tape intermediate storage.

I will use the following naming conventions. For code i, struck in Case n, I will write Cn.i. The four printing cases and the Control Case have n of 1, 2, 3, 4, and c, respectively. It is often useful, in actions designated from the Control Case, to return interpretation back to the printing cases from which Control Case was last entered -- and I designate this last-occupied printing case as Cx. Then, C1, C2, C3, or C4, used alone will designate Cases 1, 2, 3, or 4; Cc will designate Control Case; and Cn will designate any case. The form Ca, b,...,c.i will designate Code i in cases a, b,...,c.

To shift from one printing case to another requires going through the intermediate Control Case. For instance, to shift from Case 1 to Case 2, hit C1.27 (shifting to Control Case) and Cc.2 (shifting from Control Case to Case 2). Hitting Cn.27 will always ensure being in Control Case.

1. Printing Cases

Abbreviations of sorts are included. Codes C1,2.17 represent the character pair "qu" and "QU." If a "q" is desired without the "u," use Codes C3,4.26. Also, in Cases 1 and 2, Codes 29 and 30 designate more than "comma" and "period." Code 29, labelled EOF for End of Phrase, will designate the character pair COMMA SPACE. To get a comma without a space following it, use Codes C3,4.29. Codes C1,2.30 labelled EOS for End of Sentence, designate the following: "period." "space," "space," "shift to Case 2 for one character and then to Case 1." To get a period use C3,4.30. If C1.17 is hit after an EOS, the result will be the character pair "Qu" heading the next sentence.

TABLE I

CODE ASSIGNMENTS FOR FIVE-KEY BINARY HANDSET

CONTROL CASE	FINGER CODE					CODE NO.	PRINTING CASE			
	4	3	2	1	T		1	2	3	4
						0				
CASE 1					X	1	a	A	+	'
CASE 2				X		2	b	B	-	"
CASE 3				X	X	3	c	C	*	
CASE 4			X			4	d	D	div.	;
Cx, underline next alphabetic character			X		X	5	e	E	=	:
Cx, underline alphabetics until Code greater than 28			X	X		6	f	F	/	?
Cx, underline alphabetics until Code 28 or 30			X	X	X	7	g	G	(	
1-ch C2, Cx		X				8	h	H	)	$\overline{\&}$
1-wd C2, Cx		X			X	9	i	I	TAB	degrees
1-ch C3, Cx		X		X		10	j	J	$	1/2
1-wd C3, Cx		X		X	X	11	k	K	¢	B.S.
1-ch C4, Cx		X	X			12	l	L	#	
1-wd C4, Cx		X	X		X	13	m	M	@	
		X	X	X		14	n	N	%	
Backspace-Delete		X	X	X	X	15	o	O		
Backspace-Delete, Cx	X					16	p	P	0	
Delete to SP, Cx	X				X	17	qu	QU	1	
Delete thru . Cx	X			X		18	r	R	2	
Delete thru TAB, Cx	X			X	X	19	s	S	3	
	X		X			20	t	T	4	
Hand Spec.	X		X		X	21	u	U	5	
	X		X	X		22	v	V	6	
	X		X	X	X	23	w	W	7	
	X	X				24	x	X	8	
	X	X			X	25	y	Y	9	
	X	X		X		26	z	Z	q	Q
Reset Control	X	X		X	X	27	CC	CC	CC	CC
EOP, Cx	X	X	X			28	CR	CR	CR	CR
	X	X	X		X	29	EOF	EOF	,	,
	X	X	X	X		30	EOS	EOS	.	.
	X	X	X	X	X	31	SP	SP	SP	SP

Note: Cn refers to case N, where N = 1,2,3, or 4.

Cc refers to Control Case.

Cx refers to Case from which last jumped to Cc.

C4.11, a Backspace code, has the same effect as backspacing a typewriter. The originally written characters past which you backspace remain to be over-printed by succeeding printing characters.

2. Control Case

Control Case contains a number of special features. Codes 1, 2, 3, and 4 designate direct transfer to the corresponding case. Codes 5, 6, and 7 are used for underlining alphabetic characters. Code 5 says "return to Cx (the case from which you most recently transferred to Control Case) and underline the next character designated." Code 6 will send you back to Cx and cause underlining of all succeeding alphabetic characters up to the next occurrence of a code of 28 or greater--essentially, underlining the next word. Code 7 sends you back to Cx and causes underlining of all alphabetic characters until an occurrence of either a Code 28 or a Code 30--for underlining section titles. For Codes Cc.6 and Cc.7, a hyphen will be assumed as a legitimate "alphabetic" character to be underlined, but a double hyphen (representing a dash) will be a terminater of the underlining action. For Codes Cc.8 through Cc.13 we designate a short one-character or one-word "visit" to a specified case before shifting back automatically to Cx. Special note: sequence Cc.8, C2.17 results in a printing sequence corresponding to C4.26, Cx.21. Similarly, a one-character visit to Code 17 of Case 1 would produce a "q" and a Code 21 from Cx. The "one-word visit" is terminated by any code equal to or greater than 27, and that code will be interpreted as though it were in Cx.

Codes Cc.15 through Cc.19 are used for deleting preceding characters. They all have the effect upon the eventual output text of backspacing a magic typewriter that erases any character in a position to which it backspaces. Cc.15 can be hit n times successively to cause n such effective backspace-deletes, each of which eats up one line space. Cc.16 is used for the last of such a string of backspace-deletes (or when you just want one) if you want to transfer automatically thereafter back to Cx. Cc.17 will atuomatically do backspace-deletes back to but not including the previous "space", and then will transfer you back to Cx. Cc.18 will automatically do backspace-deletes back to and including

(i.e., through) the previous "period," and then transfer you to Cx. Cc.19 will automatically do backspace-deletes back to and including (i.e., through) the previous "tab," and then transfer you to Cx. IMPORTANT NOTE: A delete command is not actually executed until the next printing-case code other than Code 27.

Code Cc.21, "Hand Spec," is used as part of a procedure to tell the interpretive device which hand you are going to use on the key-set. Following recognition of this code, the computer will assume that the very next input code will use either or both the thumb and index finger, but no others -- i.e., you must jump to C1, C2 or C3 on the next stroke -- and it deduces the specified hand from this stroke. Notice that getting to this code from any state can be done entirely with symmetrical codes (codes that are the same for both hands), so that it doesn't matter what hand the interpretation was assuming, or what interpretive case was being used, when you choose to use a given hand to specify the hand-interpretation made. (A Hand Spec code automatically actuates a Cc.27 code as part of its response.)

Code Cc.27 (which in other cases causes transfer to control case) leaves interpretation in control case, but clears any pending control action which the code interpreter may be waiting to complete -- e.g., any of the underline, case-visit or delete actions.

Code Cc.28, labelled EOP for End Of Paragraph, will automatically set up the output for a new paragraph. For typewriter output, present convention, this would provide the sequence equivalent to C3.30, C3.28, C3.28, C3.9, C3.27 Cc.1, C1.27, and Cc.8.

3. Future Possibilities

Below are a few salient possibilities among the many that will undoubtedly be uncovered with future study. It will generally require coordinated analysis of editorial conventions and occurrence frequencies to establish efficient transcription codes for an interpretive program of reasonable size and operating cost.

(1) Automatic Underlining--A number of odd conditions seem to arise in trying to specify how far ahead in the text you

specify a number, n. The interpreter is to observe the symbol string S composed of the next n input characters, and use S (exactly as it would use W in the original variation) to locate the renewed starting point back in the prior input.

For on-line use with CRT feedback, one need not specify n in the latter variant of the command. He could (after giving the command) merely start entering characters and watching a "place marker" until the computer had enough input to have set the marker at the desired spot, then hit two Code 27 strokes to clear control and stop the search.

(3) First Person Singular--We can rather easily incorporate the convention that a C1.9 code (lower-case i) immediately preceded and followed by space codes, is to affect the output as if it were a C2.9 code.

(4) New Sections--This concerns text blocks larger than the paragraph. More study is needed, with establishment of consistent conventions for headings to sections of different levels. Automatic format help for these headings -- e.g., capitalization and underlining -- could be obtained, as well as coordinated subsequent-paragraph indentation. A Cc Code (or set of them) could be used to designate a new section and the section type or level. This can include listings and quoted inserts.

(5) Specifying Abbreviations--The "qu," EOF, EOS, underline, visit, delete, and EOP codes all are abbreviations, and further study of standard transcription message types will undoubtedly suggest other types. This section, however, concerns abbreviations the user may wish to establish temporarily or permanently for himself.

One Control-Case code could be used to declare that you were going to establish an abbreviation form. You then enter the full form (both control and data) of the input string you wish to abbreviate, and install a unique separate input code. You follow with the unique characters (abbreviation form, F) by which you wish to enter (designate) the abbreviation in the future. Henceforth, entry of the abbreviation form will automatically cause substitution of the full form, to produce words and/or control sequences.

want printed characters to be underlined. Cc.5 through Cc.7 as designated above don't easily handle all of these. More study is required of occurrence frequency for such conditions before one could say how best to designate codes to handle them.

One new-code possibility would say "return to Cx and underline all legitimate characters until a Cc.27 code is struck." Perhaps several such would be useful, depending upon what are to be specified as legitimate characters (e.g., sometimes a space is underlined, sometimes not).

Another new-code possibility, call it Cc.j, would be used in a two-stroke sequence Cc.j, Cx.k, which is then to be followed by printing designations in Cx. Underlining will automatically occur, for all legitimate characters, up to and including the next occurrence of Code k in Case x.

(2) Automatic Delete (and Place Finding)--In some situations, e.g., involving prior case-shift, between-sentence, and between-paragraph codes that are to be corrected, it is not clear that delete commands such as Cc.15 through Cc.19 provide the best facility for deleting and relocating, especially for an off-line operator who can't see the effect of such commands as interpreted by the computer. Also, an off-line operator may lose track of recent code strokes and/or their consequences, and may need a straightforward way to relocate correctly.

A Control-Case code could be used to tell the interpreter: Observe the next word, W, that I enter and search backward until the first occurrence of W. Delete all old entry back through the old W, and, beginning with my new W entry, accept subsequent entry as normal continuation." If you got lost, you could thus go back and restart at a previous unique word where you knew you had been doing all right. This command also allows flexible specification of deletion.

A variant of this code might be preferable, where you follow the Cc command code with a digit (or, two digits) that

Another Cc code could be used to remove a given abbreviation form, and still another Cc code could be used to designate which group or groups of abbreviations you might wish to make use of -- if such would be useful. There could well be a standard set to which in some given writing place you could usefully add some of your own. Unless you remembered these local-use additions, and could control their accumulation and discard, it could be very handy to call for a fresh start with a standard group of abbreviations.

We could ask the interpreter to check every input word against a table of abbreviation forms -- or we could constrain all such forms to begin with one particular character (or one of a special set of characters) so that lookup was done for only those input words beginning with a special character -- or we could say that a special Cc code must accompany every input word that is to be interpreted as an abbreviation form.

I would think there are few enough normal words beginning with z or x to let one or both of them, as an initial character of a word, tell the interpreter to look at the abbreviation table. Using a special Cc code to designate this, where this code has no other use, would seem to be of advantage mainly if you wanted feedback if the interpreter found no such abbreviation form in its table -- since with such as the z or x designater we would have to let the interpreter print the input word if it didn't find it on the table.

(6) Control Trees--We could foresee a possibility in the foregoing section of a number of different control commands associated with setting up or deleting abbreviations. These wouldn't be too frequently used, and codes will become precious as we learn more use for them. For less-frequent control commands, it could pay to bundle them in one or more second-level control cases to which entry is made from the first-level Control Case (the one we now have). For still-less-frequent commands, third-level control cases might be fitting, etc.

Alternating two hands on two keysets definitely seems a useful possibility for faster transcription. The interpreter has to keep track of which keyset provides a given entry, and a separate Hand Spec procedure is needed for each keyset. Otherwise the string of sequential input codes is interpreted without regard for which keyset enters which code.

We can stipulate that the user, for this type of interpretation, never overlaps his key strokes -- i.e., only one set of keys is depressed at a time. We can then stipulate that if overlap exists, the two overlapped codes are to be interpreted differently.

For instance, we may stipulate that a code stroke that is held so as to be overlapped by the succeeding other-hand stroke is to be interpreted as a Cc code -- with the succeeding overlap code being interpreted normally. But this only saves one stroke, and at the expense of a break in stroke rythmn.

A better use of two-hand overlap would seem to stem from saying that the two overlapped five-bit codes are to be interpreted as an independent ten-bit code. This gives us 961 unique codes with which to designate special abbreviations, control tricks, etc. and opens the door to a coordinated "shorthand" code. Phrases, words, n-grams (including spaces, punctuation, and control characters), whose payoff in frequency of use times 5-bit code strokes saved is large enough, would thus be encoded as single two-handed overlap strokes.

It seems reasonable to say that it is the order of keyset actuation, rather than which hand actuates which code, that should determine the way the interpreter groups the two five-bit codes into a ten-bit code. This would seem less disturbing to the character-at-a-time alternate-hand rhythm in which the shorthand code would be embedded.

However, it would be possible for the order of overlapping -- which hand was first -- to be of significance. This could provide two independent 961-code interpretive repertoires. It is also possible to have the interpretation give the same designator role to the same hand, no matter which of the overlapped pair was struck first.

The shorthand forms can be remembered and stroked as a pair of C1 characters rather than as a ten-bit code. Also, this shorthand springs compatibly from the five-bit character-by-character code, and a person can gradually add new shorthand forms to his repertoire as his familiarity and skill grow.