ESD TDR 64-370
ESTI FILE COPY

ESD-TDR-64-370

ESTI PROCESSED
☐ DDC TAB ☐ PROJ OFFICER
☐ ACCESSION MASTER FILE
☐ _____
DATE _____
ESTI CONTROL NR AL 42184
CY NR 1 OF 1 CYS

ESD RECORD COPY
RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTI), BUILDING 1211

COPY NR. _____ OF _____ COPIES

Group Report 1964-51

D. B. Yntema

The Software Problem

4 September 1964

Prepared under Electronic Systems Division Contract AF 19(628)-500 by

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Lexington, Massachusetts

AD605824

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the U.S. Air Force under Contract AF 19(628)-500.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

THE SOFTWARE PROBLEM

D. B. YNTEMA

Group 25

GROUP REPORT 1964-51

4 SEPTEMBER 1964

LEXINGTON MASSACHUSETTS

ABSTRACT

The question of making digital computers more useful to technical personnel like engineers and scientists is discussed informally. It is suggested that computing systems should be designed for a type of use described as "step-display-look." It is also suggested that once the user has been put on-line, the problem of software becomes critical. In particular, the clerical labor that is usually required in instructing the computer becomes an important obstacle to rapid interaction between man and machine. The requirements for an experimental system intended to put these opinions into practice are sketched, and some of the major decisions that have been made in planning such a system are discussed briefly.

Note: This report is a talk given to the General Research Panel of the Lincoln Laboratory Joint Advisory Committee on 9 April 1964. The thoughts and plans discussed here were the result of a collaboration between members of Group 25 (Psychology) and Group 23 (Digital Computers).

Accepted for the Air Force
Franklin C. Hudson, Deputy Chief
Air Force Lincoln Laboratory Office

For the last few months we have been considering the problem of computer software, especially software for technical applications like science and engineering. I shall describe the opinions that we have formed, and then say a little about an experimental system on which we are working in an attempt to put our opinions into practice.

The classical picture of the way one uses a computer is shown in Fig. 1. The process starts with a question; the question is refined into a specification of the problem; the specification is elaborated into flow charts; the flow charts are translated into a coded program, which is debugged to correct the inevitable errors; and finally the completed program is run.

Question → Problem Specification → Flow charts → Code and Debug → Run

Figure 1

When you consider how you would like to see a scientist or engineer using a computer, the picture that emerges is quite different – more like that shown in Fig. 2. Typically the user begins with a set of data on which he thinks it would be revealing to perform some sort of calculation; or as we

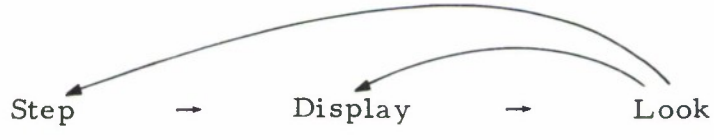

Figure 2

put it, he would like to make some sort of computational step. When he has made the step he will want to see what he has done; so he will display the results and look at them. After studying the display he may want to see a different aspect of the results – perhaps he will want a graph with a different scale – and he may go around the smaller loop several times, looking, getting a new display, and looking again. Sooner or later he is likely to feel that it would be instructive to do some other computation – i. e., make a further transformation on the results of the previous step, or go back to the original data and do something entirely different. In either case he will want to get a new display, look at what has happened, and so on, round and round.

The word "step" is used here in a very special sense. As we define it, a step consists of whatever computation the user wants done between times when he feels compelled to look at the results. Thus the size of a step is likely to change as a person works on a problem. At the beginning he will probably take small steps, looking each time to see what has happened. But after he begins to understand the problem he may want to run off a whole series of small steps before bothering to see what he has done. When that occurs, the series of small steps has by our definition fused into a single large one.

The word "look" is also used in a special sense. During this interval the person is not only staring at the display; he may also be considering his next step, looking for regularities in the data, rethinking his plans for doing the computation, or perhaps, on rare occasions, having the flash of insight that restructures his whole conception of the problem. One could say that it is during the time labeled "look" that the user does the sort of work for which scientists and engineers are hired. It is during this time that the man in the system earns his keep.

Let me make a couple of remarks about these two diagrams. The "step-display-look" procedure is intuitively appealing, but it is not an efficient way to use a conventional installation where you wait three or four hours to get answers after you submit a job to be run on the machine. If

you proceeded a step at a time, looking after each step, you could make about two steps a day. It would take weeks at that rate to do even a moderate-sized job of computation.

So in practice you put a number of steps together and submit them to be done on the computer in a single run – in other words, you work in the classical manner shown in Fig. 1. The trouble with trying to take a lot of steps in one run is that you must exercise a painful amount of foresight. For example, if you think it would be instructive to take the logarithms of a set of measurements, and if the measurements are subject to enough error so that some of them could be negative and have no logarithms, you must anticipate that difficulty and tell the computer what it is to do if it encounters a negative number. If you do not, the answers you get may be flatly wrong, or, more likely, the program may stop in a disorderly jumble that will cost you hours of detective work to sort out. You are not only required to foresee the direction in which you expect the computations to go; you are also required to anticipate all of the other things that could happen but probably will not. You must tell the machine what to do in each of these special cases, or at least tell it how to stop gracefully. This is a lot of extra work.

To take a small example, one of our staff has written a routine for doing floating-point addition on the TX-2 computer. He says that a little over two-thirds of the instructions in the routine were included just to take care of special cases – things that might happen but usually do not. He also says that writing those instructions represented a good deal more than two-thirds of the labor expended in writing the routine. This is probably an extreme example, but it shows how expensive the necessity for foresight can be.

There is also a larger, more subtle sense in which foresight can be difficult or impossible. In dealing with actual data it is often hard to say in advance exactly what calculations you will want to make. As the results begin to emerge you notice an unexpected pattern in the numbers, or you plot a set of curves and discover that one of them shoots off at an odd angle;

so you feel constrained to do additional calculations to find out why. This sort of occurrence seems to be more the rule than the exception. I doubt that I have ever seen a data-analysis project – at least, not one involving more than a few dozen numbers – that went exactly according to the preconceived plan. The natural and efficient way to work with data is therefore to proceed a step at a time in the fashion symbolized by Fig. 2.

Let us assume that we are now moving into the era of logic plenty, the time when computing power will be cheap – as it is already beginning to be. In that case we should be starting to design computer systems that will make optimal use of people, not optimal use of the machines. If a person works most efficiently when he proceeds a step at a time, then our first task is to reduce drastically the interval between times when he can "look." It simply is not practical to take a single step and then wait for hours to see the result. Our second task, once we have gotten to the point where the "step-display-look" process is feasible, is to make that process efficient. What we have to do is obvious. The man earns his keep during the time labeled "look"; so our job is to minimize the time he must spend on the other two parts of Fig. 2, taking a step and getting a display. In a way the two tasks really come down to the same thing. If we want to make optimal use of the man we must minimize the time between his "looks."

The time between looks can be considered in two parts. First, there is the time required for the man to give his instructions to the computer. He has to say what step if any he wants to take and what display he wants to see. Second, there is the time he must wait for a response to the instructions he has given.

I shall not say much about reducing the wait for a response. That problem has received a great deal of attention in the last two years, and the place to begin is fairly obvious. The first step is to put the user on-line – for example, give him a typewriter connected to the computer so that he can get a response in minutes, maybe seconds, instead of waiting three or four hours. The advantages are very familiar to us here at Lincoln. Our TX-2

computer was designed for on-line use, and we have been using it that way for about five years. More recently we have had a couple of smaller machines that were designed to be used in the same way, and now we are fortunate in having two teletypes connected to the computer down on the campus at Project MAC, which has a multi-user, time-shared installation.

When you have worked on-line and have discovered what it feels like to get a response in a few seconds, you become very conscious of the other factor, the time required to tell the machine what you want. With those teletypes you often get a great deal of computing done in, say, fifteen seconds after you press the button that tells the machine to go. But before you press the button, you have probably taken at least fifteen minutes to think out and type the instructions that tell the machine what to do. There is a bad mismatch here. Once the user is on-line, the time he spends instructing the machine is much longer than the time he spends waiting for his work to be done. The ratio is ten to one at the very least – more likely a hundred to one.

We feel that this mismatch of perhaps two orders of magnitude constitutes the software problem. Or, strictly speaking, it constitutes the problem in the use of computers by technical personnel. Software may pose other problems in other fields, but in the present state of the art the ratio of instruction time to response time is the primary difficulty in the use of computers to answer technical questions.

Why should it take so long for the user to tell the machine what he wants it to do? For one thing, he spends a lot of time on plain, stupid clerical work – labor of the sort that is particularly painful to see a person doing because people do it so badly and computers do it so well.

Let me give some examples to remind you just how burdensome the clerical labor can be, even when you are using a language like FORTRAN. *

---

* In the discussion that follows I often use FORTRAN and the Project MAC teletypes as illustrations. Since I discuss ways in which the use of computers should be improved, it is conceivable that some readers might misconstrue my remarks as disapproval of FORTRAN or of the Compatible Time-Sharing System used by Project MAC. In committing this discussion to paper I wish to make it quite plain that if anyone thinks I am disparaging two such important accomplishments, he has missed the point entirely.

Suppose you want to multiply two matrices together: matrix A is to be multiplied by matrix B and the result is to be called matrix C. Figure 3 shows what you have to do. You must have remembered the dimensions of A and B, or have jotted down a reminder like the note in the upper right-hand corner of the figure. The six lines of FORTRAN then say that A is to be post-multiplied by B and the result is to be called C.

A is a 25 x 11 matrix.
B is a 11 x 4 matrix.

```
        DO    152     I  =  1, 25
        DO    152     K  =  1, 4
        C(I, K)  =  0.
        DO    152     J  =  1, 11
152     C(I, K)  =  C(I, K)  +  A(I, J)*B(J, K)
        DIMENSION  C(25, 4)
```

Figure 3

There are two things to notice here. First, the time required to type these six lines is not trivial (an average of a little over two minutes for six users of the teletypes that I mentioned). Second, there are quite a few things that have to be checked against each other. The so-called statement number, 152, must match in the four places in which it appears, and you must check that there is no other statement numbered 152 anywhere else in the program. Note too that the dimensions, 25, 4, and 11, must be typed (even twice) and must check against the dimensions in your reminder. In effect you are telling the machine something that you have told it already — namely, the dimensions of A and B — and something that it should be able to

figure out for itself – namely, the dimensions that C is going to have.

The problem of clerical labor can be avoided to some extent by using a prepackaged subroutine, either a routine that you have written yourself or one that you have found in a library. This is not a total solution; a certain amount of clerical work is still required in meshing the routine into your program. As an example, Fig. 4 shows the use of a routine for finding the latent roots and vectors of a positive semi-definite symmetric matrix.

E is a 12 x 12 matrix.

```
N = 12
IGEN = 0
CALL  HDIAG (E, N, IGEN, U, NR)
DIMENSION  U(12, 12)
```

Inside the routine:

```
DIMENSION  H(12, 12), U(12, 12), X (12), IQ (12)
```

Figure 4

There are four lines of typing that you have to include in your program. Even worse, you must go down into the routine itself and change a line, as shown at the bottom of the figure. And note that you have to tell the machine over and over – nine times to be exact – that this is a 12 x 12 matrix. (FORTRAN programmers will realize that there is a way in which one can avoid making the dimensions inside the routine correspond exactly to the dimensions of the matrix, but that raises other problems.)

Let me take one more example. Suppose you have computed the matrix C that I described a moment ago, and now you look at it and decide

to perform some further calculation on it. Figure 5 shows the easiest way I know of doing it on the teletype. The two statements shown at the top of the figure are included in the program in which C was originally computed (note that you had to foresee that you might want to use C again) and the new program begins with the three lines shown at the bottom. Again there are a number of things that have to check: the tape number, 24; the dimensions, 25 and 4; the format, 6O12; and so on. (FORTRAN programmers may be surprised at the tangles of parentheses that appear in the input and output statements. The version of FORTRAN available on the teletype apparently requires that you do it this way.)

In old program:

```
        WRITE OUTPUT TAPE 24, 137, ((C(I, J), I = 1, 25), J = 1,4)
137     FORMAT (6O12)
```

In new program:

```
        DIMENSION C(25, 4)
        READ INPUT TAPE 24, 1, ((C (I, J), I = 1, 25), J = 1, 4)
  1     FORMAT (6O12)
```

Figure 5

It is worth a moment to mention that the clerical gymnastics shown in Fig. 5 are not required when you want to save a program – only when you want to save a file of results. The system to which the teletypes are connected was designed, like most current on-line systems, as a tool for programmers to use in debugging programs. The designers provided a delightfully convenient way of retaining the things in which a programmer is

most interested, the various versions of his program. To save a new version you type a single word followed by the name by which you want the version to be called; then whenever you want it again you need only type its name and one other word. We assume that a man doing computations would be equally delighted to have the same sort of convenience in retaining the things that are of primary interest to him, namely, the files of numbers on which he is working.

What can be done about the problem of clerical labor? The only solution we can think of is the obvious one – use a prepackaged routine when you want to make a step or get a display. The system may include a library of routines that perform the desired actions, or it may include routines that will on demand put together a program to perform an action; for the moment the difference is not crucial. Then too, the routines may be ones that some public-spirited person has deposited in a library, or they may be ones that the user has written for himself. Occasionally the user may even stop work and write a new routine. That is not crucial either, provided he does not have to write a new routine too often. The crucial point is this: If routines are used to make steps and get displays, then during every interval between looks the user will go through the process of applying at least one routine. Thus it is important that applying a routine should not involve the sort of clerical labor that was illustrated in Fig. 4.

This is the place where innovation is needed. There is seldom anything mysterious about routines that do the sort of calculations and display the sort of graphs that engineers and scientists want. Building up a library of such routines may be a lot of work, but in the present state of the art it is usually a straightforward job. What is lacking is a convenient way of applying the routines.*

---

* A notable exception is the system designed by Glen J. Culler and Burton D. Fried. See their report, An on-line computing center for scientific problems, M19-3U3, revised June 1963. TRW Computer Division, Thompson Ramo Wooldridge Inc., Canoga Park, Calif.

The best way to summarize what I have said so far is to list what we consider the minimal requirements for an experimental system that will make it feasible to work in the "step-display-look" fashion represented by Fig. 2. In particular, the objective is to reduce the interval between "looks" by putting the user on-line and by holding the clerical labor down to some reasonable level.

1. Fast response by the machine. Obviously we should make the machine reply as quickly as we can manage.
2. Retention of results. Unless the user deliberately erases the results of a step they should be preserved in such a form that they can be used as the basis for further calculations on which he may decide later.
3. Minimum clerical labor in applying a routine. We have assumed that the user will normally take steps and get displays by applying routines that he or someone else has worked out beforehand; so we are particularly anxious to minimize the labor of calling a routine.
4. Ease in combining steps into larger steps. We want to make it easy for a person to talk in units that he regards as steps. When he progresses to thinking in larger units it should be easy for him to combine a series of small steps into a single big one.
5. Moderate ease in adding a new routine. The easier it is to write new routines, the more of them we will be able to provide in the library. And when the library is insufficient, the user will have to write his own. For both reasons there must be no special obstacles to making up a new routine and fitting it into the system.

6. Leisure to look. As before, "look" is used to cover all the activities by which the user earns his keep. If he is to do any serious thinking he must have the leisure to sit and ponder; he must have access to the computer for good long periods of time.

Obviously this list does not cover the entire problem of software for the use of computers in answering technical questions. Let me emphasize that point by mentioning three omissions of which we at Lincoln have particular reason to be conscious.

First, technical men often like to communicate with each other by drawing pictures and graphs; sometimes they would like to communicate with the computer in the same way. We are not likely to forget that fact; work on graphical communication from man to machine has been going on here for several years.

Second, many technical people do not type well; they would much rather talk to the machine. We are conscious of that too; a project on computer recognition of speech has been underway here for some time.

Third, there are obvious advantages in letting a person use the language and symbols to which he has become accustomed in the field in which he is working. In other words, it is desirable to use what is often called a problem-oriented language. We have a comparatively new research project on a very general translator that will allow people to define symbols and even specify syntax on-line. We think this translator may prove convenient enough so that a person working on a technical problem may be able to create his own problem-oriented language as he goes along.

So we realize that our list of requirements is not exhaustive. It concentrates on what we regard as the big, obvious problem – clerical labor. We are following the strategy often adopted in adjusting a piece of apparatus or in debugging a program. You don't try to do everything at once. If you see some large, obvious trouble that you think you can fix, you deal with it first, and then the remaining problems may be easier to understand.

We are currently trying to build a system that will satisfy the requirements that I listed. We are using TX-2 as the computer, the Lincoln Writer keyboard as the primary input device, and the scope and Lincoln Writer printer as the primary outputs.

Suppose again that the user wants to multiply matrix A by matrix B and call the result C. He types something like "MATMUL A B C," and the machine performs the multiplication, taking care of all the clerical details. To be more explicit, the statement "MATMUL A B C" is put in a sort of temporary storage. The executive system inspects the first item, "MATMUL," finds it is the name of the matrix-multiplication routine, puts that routine in the location in which it was designed to run, and transfers control to it. The routine itself fetches the second name in the statement and asks the executive to put the file having that name in some location that the routine specifies. The file having the third name, B, is treated in the same way. The routine then consults headers on files A and B to see whether they are indeed matrices that can be multiplied together. If they are, it determines what the dimensions of the resulting matrix will be, fetches from the statement the fourth name, C, and tells the executive to create a file called C that will be large enough to hold the new matrix. Finally the routine performs the multiplication and puts on the new matrix a header showing its dimensions. The matrix C is then ready to be used as the input to any further operations on which the user may decide. *

The statement "MATMUL A B C" is not very pretty to look at; but as I said, problem-oriented languages are not on our list of minimal requirements. On the other hand, we are including two features that should go a long way toward making the system convenient to use. First, there is the creation of synonyms. If the user is doing lots of matrix multiplication and

---

* The sequence of events described above is correct in spirit but not in detail. The plans have been changed slightly since this talk was presented.

gets tired of typing "MATMUL," he can decree that some shorter expression like "MM" shall be synonymous to "MATMUL." Thereafter he can just type "MM" and the system will understand that the matrix multiplication routine is what he wants. Second, there is the concatenation of steps. If the user finds that he is typing the same sequence of four or five statements over and over, he can decree that the whole sequence of statements shall have a single name. Then whenever he types that one name it will be as if he had typed all four or five lines. In other words, he can combine a series of small steps into a larger one.

It is not my task to describe the system in detail, but I shall conclude by listing five of the major decisions we have made in designing it:

1. Time-sharing. We have decided to time-share the computer so that several people can use it at once. This decision does not imply any judgment about the ultimate merits of time-sharing. The computer we have to work with is TX-2, which is already in heavy use. If we give the machine to one person at a time he will not have much leisure to sit and think.

2. Routines are files that operate on files to create files. This heading really summarizes several decisions. We suppose that the library of routines will be too big to keep in core all at once; it must be broken into pieces that can be brought from an auxiliary memory when they are needed. Typically each piece will be a routine, and in that sense we say that a routine is normally treated as a file. Similarly, when a person works on a problem for a while he is likely to accumulate a mass of data and results too large to keep in core; the mass of information must again be broken into units that can be handled separately. Blocks of data or of results are therefore treated as files: each block to which the user has given a name is handled as a unit when it must be brought from the auxiliary memory. The system makes no essential distinction between files of data and files of results. The results of one routine may be used later as input to another routine; so inputs and results must look alike.

3. Clerical labor is done mainly by the library of routines, not by the executive system. The executive maintains an index of the user's files, but

the main burden of the clerical labor, the labor we are trying to take off the user's shoulders, falls on the individual routines that he calls from the library.

This is a rather interesting point. We started with the idea that the executive system would assist the library routines by checking their inputs to see whether the operation was legal, by preparing the header to put on the results, and so on. But as we proceeded, it became clear that these clerical functions would be different for almost every new routine. To keep the executive from expanding indefinitely we decided to take these functions out of the executive and put them into the individual routines that they served.

4. Routines are in absolute binary. They are in binary so as not to lose time compiling or interpreting them every time they are used. In absolute, because relocation registers looked faster and simpler than relocatable routines.

5. Hardware to shuffle addresses. When the executive brings a file from auxiliary storage it may have to move other files around to make space for the new one. There are in principle two ways of moving information from one address to another. First, you can copy the information from the magnetic cores at one address into the cores at the other address. Copying can take a lot of time if there are many registers of information to be moved. Second, you can leave the information in the register where you found it, but change the address of the register. We have chosen the second course. We are modifying the memory-address hardware on TX-2 so that blocks of 256 registers may seemingly be moved very rapidly to any of a thousand positions in core memory.

And we have made one further decision: as you may have guessed from the examples that I have used, the first set of library routines that we plan to provide are routines for matrix arithmetic.

To sum up, the fashion in which one would like to see technical people working with computers in the era of cheap logic is the fashion that I have described as "step-display-look." When the response time has been cut down to a reasonable level, the big remaining problem is that of software, the

problem of telling the machine what you want it to do. One important component of the software problem is the large amount of clerical labor demanded of the user, labor of the kind that people do badly and computers do well. We think it possible to take much of this labor off the person's shoulders. We shall attempt to do so, and then the remaining problems may be easier to understand.

14. KEY WORDS	LINK A ROLE	LINK A WT	LINK B ROLE	LINK B WT	LINK C ROLE	LINK C WT
Software						
Step-Display-Look						
Computer						
Programmers						
Teletype						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Lincoln Labs., Lexington, Mass.

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

N/A

3. REPORT TITLE

The Software Problem

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Group Report

5. AUTHOR(S) *(Last name, first name, initial)*

Yntema, D.B.

6. REPORT DATE

Sep 64

7a. TOTAL NO. OF PAGES

17

7b. NO. OF REFS

0

8a. CONTRACT OR GRANT NO.

AF19(628)500

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

GR-1964-51

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

ESD-TDR-64-370

10. AVAILABILITY/LIMITATION NOTICES

Qualified Requesters May Obtain Copies From DDC.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

ESD, L.G. Hanscom Field, Bedford, Mass.

13. ABSTRACT

The question of making digital computers more useful to technical personnel like engineerings and scientists is discussed informally. It is suggested that computing systems should be designed for a type of use described as "step-display-look." It is also suggested that once the user has been put on-line, the problem of software becomes a critical, In particular, the clerical labor that is usually required in instructing the computer becomes an important obstacle to rapid interaction between man and machine. The requirements for an experimental system intended to put these opinions into practices are sketched, and some of the major decisions that have been made in planning such a system are discussed briefly.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification