N73-19220

PROGRAM AUTOMATED DOCUMENTATION METHODS

Bernadine C. Lanzano
TRW Systems Group

Several methods for automatically generating and maintaining documentation for TRW's computer programs are being used, and other procedures are under examination. This paper presents a short synopsis of the Mission Analysis and Trajectory Simulation (MATS) program to provide an understanding of the size and complexity of one simulation for which documentation is mandatory, a description of a program that assists in automating the documentation of subroutines, an exposition of two flowcharting programs, some notes on useful program internal cross-reference information, an implementation of a text-editing program available in a time-shared computer system environment, a preview of a proposed system that would aid in program development and documentation that utilizes a graphics display console, and a recommendation for software standardization.

In the complex world of sophisticated computer systems and advanced software technology, Thompson Ramo Wooldridge. Inc. (TRW), has long recognized the need for developing general-purpose programs, automating documentation, and standardizing programming techniques. The satisfaction of these demands minimizes software expenses by eliminating program duplication, developing new capabilities around and within existing programs, responding in a quick reaction real-time sense, and by generating documentation with minimum effort.

The MATS program is briefly mentioned because its generality, complexity, and size necessitate considerable support documentation. Because this program is used by a variety of projects, its documentation is referenced by many engineers, programmer/analysts, and technical aids. Methods of automating the documentation were deemed mandatory.

MATS is a digital computer program that simulates ballistic and space mission trajectories; it either has or is capable of simulating such missions as Apollo, Pioneer, Minuteman, and Grand Tour, with such vehicle configurations as Saturn, Atlas, Titan, Agena, Centaur, Minuteman, and Thor, among others. The program is written in FORTRAN IV and is operable on the CDC 6000, GE 635, IBM 360, and IBM 7094 computer systems. MATS is composed of more than 450 subroutines that occupy some 110 000 decimal words in 15 overlaid segments. The program control logic is predicated on a modular design concept that facilitates the addition or exchange of capabilities for the various missions. It can be mated with control systems that include navigation and guidance algorithms and can provide the dynamics for interpretive computer simulation systems.

For any program, several levels of documentation are required. The smallest unit is the subroutine where the function, algorithms, and data communication must be explained. A lineal linkage trace of the program logic control hierarchy provides the next level of documentation. Two-dimensional cross-reference information is desirable so that not only which subroutine(s) and common storage(s) are used by a given routine are known but also which routines reference this routine and which routines reference each element of global storage.

The descriptive material that functionally relates the program modules can be generated only by the program architect and must be updated as the program expands and evolves. Last but probably most important are the user manuals, which again must be generated in an easily maintainable format.

This paper describes several auxiliary programs that support the automatic documentation of MATS and other programs.

(1) Automated Documentation of Subroutines (ADS) mechanizes the descriptive explanation at the subroutine level.
(2) Flowcharters AUTOFLOW and FLOWGEN pictorialize the computational algorithms and decision branches within a subroutine. AUTOFLOW can be requested to provide a higher level flowchart.
(3) Automatic Flow Layout of Program (AFLOP) maps the subroutine usage at the program level.
(4) Methods for retrieving and programs for generating the cross-reference information are presented.
(5) Administrative Terminal System (ATS) assists in automating the maintenance of handbooks and manuals.
(6) Computer-Aided Program Development (CAPD) proposes a method wherein the coding and final flowcharting no longer appear as steps in program development and documentation.

One of the purposes of this paper is to demonstrate how an individual piece of information can be made to function in more than one capacity, thus deleting duplication of effort in creating and maintaining documentation files.

AUTOMATED DOCUMENTATION OF SUBROUTINES

This program, currently under development, accepts as inputs FORTRAN subroutine source decks, a subroutine titles file, and a symbols definitions file. An option to retrieve storage and external reference information from the compiler output or the list tape is under review.

The source deck contains the following information, where Cnn is the appropriate coded comment card: the title, a one-line title that normally spells out the name of the subroutine (C10); the author, the programmer/analyst, and other personnel cognizant of the function and/or implementation of the subroutine (C30); the abstract, a meaningful description of the purpose and function of the subroutine (C40); remarks, information that briefly describes any change and gives the analyst's name and date and any special features, restrictions, limitations, and error treatments (C50); and the local variables, the names and definitions of those variables that are used only within the subroutine including those in the calling sequence (C60).

The subroutine title file is merely the title (C10) cards identified with their respective subroutine names.

The symbols definition file contains the name, common block location, definition, and units of every global variable used in the program.

These cards are formatted in the following manner:

```
C EQU (VARNAM, COMBLK(nnnn))   DIM(iii,jjj)  IO  UNITS=kkkk      VARNAM  00
        DEFINITION (on as many cards as required)                VARNAM  mm
```

where EQU implies equivalence; VARNAM is the variable name; COMBLK(nnnn) is the name of the common array and location of the variable within the block; DIM(iii,jjj) is the dimension of the variable; I or O indicates whether it is a user input or a program computed variable; UNITS = kkkk provides information pertinent to the variable, such as length and time units, integer or real format, and special processing information; and DEFINITION continues from one card to the next and is identified by the variable name and card count mm in the final columns.

A parenthetical explanation of the symbols definitions file is in order. The MATS program requires the information provided by the EQU cards for the symbol table, input, computations, and output processors to locate and identify each variable so that all input and output may be recognized by symbol name. The symbol table processor accepts as input the EQU . . . 00 file, alters the file by change directives, and outputs an updated file. The symbol definitions file, using standard sorting equipment, is updated and published as a portion of the MATS users manual. This file, being carefully designed and formatted, thus may be used in three distinct areas: the MATS program, the ADS program, and the MATS documentation. Furthermore, it is easily and automatically updated.

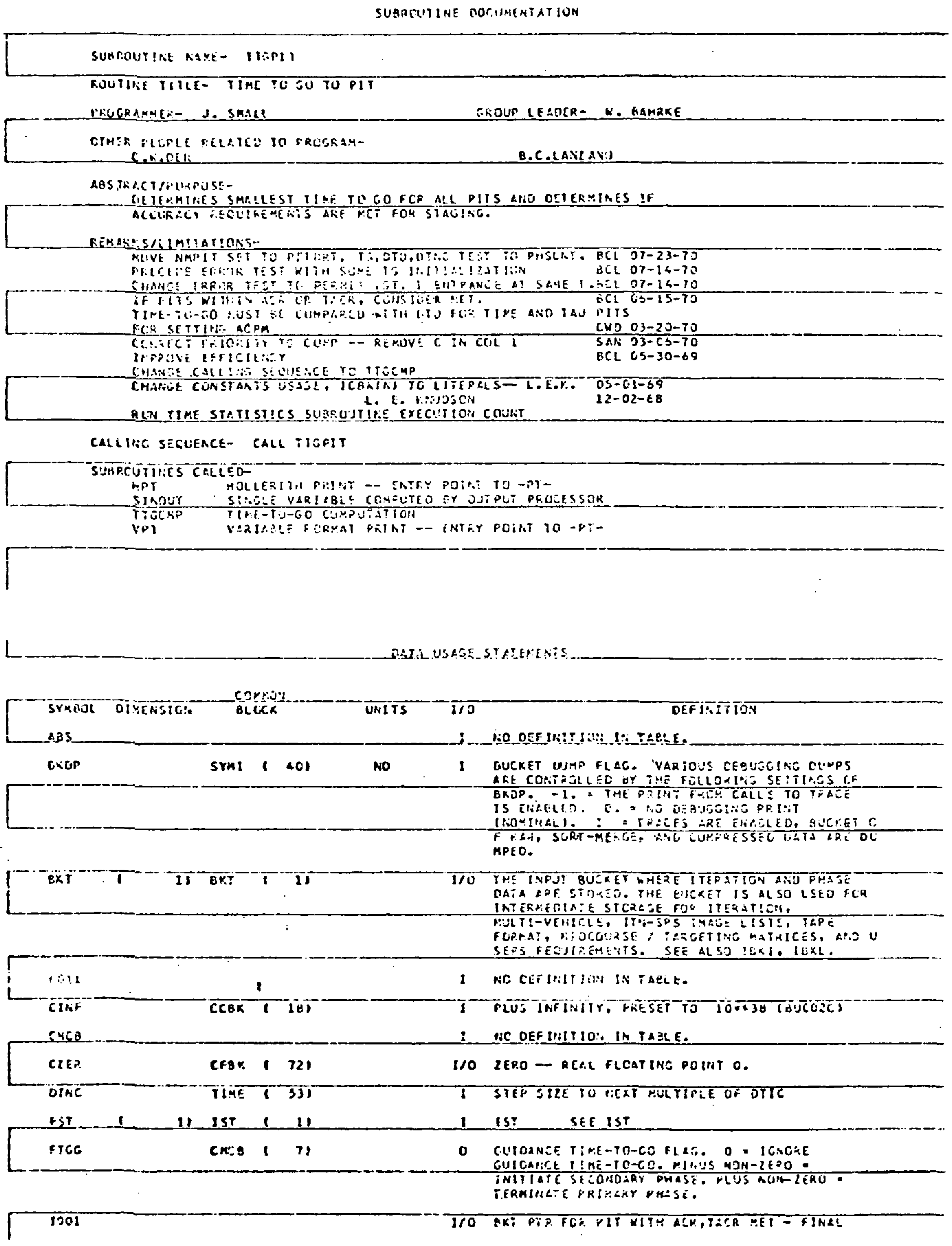
The ADS program searches the source deck from the subroutine card to the end card. It retrieves the name from the subroutine card and the title, author, abstract, and update information from the coded comment cards. It identifies the external references and acquires their titles from the titles file. It identifies the variables and locates their definitions either from the symbol definitions file or from the C60 cards. Figure 1 presents the output of the ADS program.

The appearance of the coded comment cards in the MATS program listing proves very useful to the programmer/analyst who, when working with the subroutine, finds the associated documentation immediately available.

TRW has other programs similar to ADS that operate on the IBM 7094 and IBM 360 to document programs that execute on those computer systems. Each is clearly designed to automate subroutine documentation with minimum manual effort.

FLOWCHARTING PROGRAMS

Two flowcharting programs are currently in use at TRW: AUTOFLOW and FLOWGEN. Each is leased from the respective vendor. They are used primarily for the charting of individual subroutines, but AUTOFLOW can produce charts for a complete (small) program. The symbology of both is nearly self-explanatory and quite similar to that commonly used by analysts.



```
                                   SUBROUTINE DOCUMENTATION

        SUBROUTINE NAME-  TTGPIT

        ROUTINE TITLE-  TIME TO GO TO PIT

        PROGRAMMER-  J. SMALL                         GROUP LEADER-  W. BAHRKE

        OTHER PEOPLE RELATED TO PROGRAM-
             C.W.DEN                                  B.C.LANZAND

        ABSTRACT/PURPOSE-
             DETERMINES SMALLEST TIME TO GO FOR ALL PITS AND DETERMINES IF
             ACCURACY REQUIREMENTS ARE MET FOR STAGING.

        REMARKS/LIMITATIONS-
             MOVE NMPIT SET TO PITBRT. TS,DTO,DTNC TEST TO PHSCNT. BCL 07-23-70
             PRECEDE ERROR TEST WITH SOME TS INITIALIZATION       BCL 07-14-70
             CHANGE ERROR TEST TO PERMIT .GT. 1 ENTRANCE AT SAME T.BCL 07-14-70
             IF PITS WITHIN ACR OR TACR, CONSIDER MET.            BCL 05-15-70
             TIME-TO-GO MUST BE COMPARED WITH DTO FOR TIME AND TAU PITS
             FOR SETTING ACPM                                     CWD 03-20-70
             CORRECT PRIORITY TO COMP -- REMOVE C IN COL 1        SAN 03-05-70
             IMPROVE EFFICIENCY                                   BCL 05-30-69
             CHANGE CALLING SEQUENCE TO TTGCMP
             CHANGE CONSTANTS USAGE, ICBKIN) TO LITERALS-- L.E.K.  05-01-69
                                  L. E. KNUDSON                    12-02-68
             RUN TIME STATISTICS SUBROUTINE EXECUTION COUNT

        CALLING SEQUENCE-  CALL TTGPIT

        SUBROUTINES CALLED-
             HPT        HOLLERITH PRINT -- ENTRY POINT TO -PT-
             SINOUT     SINGLE VARIABLE COMPUTED BY OUTPUT PROCESSOR
             TTGCMP     TIME-TO-GO COMPUTATION
             VPT        VARIABLE FORMAT PRINT -- ENTRY POINT TO -PT-




                                  DATA USAGE STATEMENTS


                        COMMON
    SYMBOL  DIMENSION    BLOCK           UNITS     I/O                    DEFINITION

    ABS                                             I   NO DEFINITION IN TABLE.

    BKDP                SYMI  (  40)      NO        I   BUCKET DUMP FLAG. VARIOUS DEBUGGING DUMPS
                                                        ARE CONTROLLED BY THE FOLLOWING SETTINGS OF
                                                        BKDP.  -1. = THE PRINT FROM CALLS TO TRACE
                                                        IS ENABLED.  0. = NO DEBUGGING PRINT
                                                        (NOMINAL).  1  = TRACES ARE ENABLED, BUCKET O
                                                        F MAH, SORT-MERGE, AND COMPRESSED DATA ARE DU
                                                        MPED.

    BKT     (      1)   BKT   (   1)               I/O  THE INPUT BUCKET WHERE ITERATION AND PHASE
                                                        DATA ARE STORED. THE BUCKET IS ALSO USED FOR
                                                        INTERMEDIATE STORAGE FOR ITERATION,
                                                        MULTI-VEHICLES, ITN-SPS IMAGE LISTS, TAPE
                                                        FORMAT, MIDCOURSE / TARGETING MATRICES, AND U
                                                        SERS REQUIREMENTS.  SEE ALSO IBKI, IBKL.

    [illegible]                                     I   NO DEFINITION IN TABLE.

    CINF                CCBK  (  18)                I   PLUS INFINITY, PRESET TO  10**38 (BUC02C)

    CMCB                                            I   NO DEFINITION IN TABLE.

    CZER                CFBK  (  72)               I/O  ZERO -- REAL FLOATING POINT 0.

    DTNC                TIME  (  53)                I   STEP SIZE TO NEXT MULTIPLE OF DTIC

    FST     (      1)   IST   (   1)                I   IST       SEE IST

    FTGG                CMCB  (   7)                O   GUIDANCE TIME-TO-GO FLAG.  0 = IGNORE
                                                        GUIDANCE TIME-TO-GO. MINUS NON-ZERO =
                                                        INITIATE SECONDARY PHASE. PLUS NON-ZERO =
                                                        TERMINATE PRIMARY PHASE.

    I001                                           I/O  BKT PTR FOR PIT WITH ACR,TACR MET - FINAL
```

Figure 1.–Example of ADS output.

AUTOFLOW accepts COBOL, FORTRAN, IBM 360 Assembly, and PL/I source programs as inputs. The AUTOFLOW option generates a chart set composed of the title sheet, input listing, statement label index, table of contents, table of diagnostics, flowcharts, and other special listings; some of these items are optional. It charts an entire program up to 999 flowchart pages.

The CHART option operates from specially coded comment cards that may be embedded in the program source deck and produces a higher level program chart from the textual information. The author may adjust the level of detail to the type of chart he wishes to exhibit.

AUTOFLOW executes on the IBM 360; each chart page covers two 11- X 17-in. printer pages and may contain up to four columns of paths. Pages and symbols are numbered to facilitate page-wide logic flows. Figure 2 exhibits an AUTOFLOW chart.

FLOWGEN accepts FORTRAN source decks as inputs and outputs a chart somewhat less sophisticated than AUTOFLOW. No provision exists for a level of detail control. It charts individual subroutines.

FLOWGEN executes on the CDC 6000 and generates input for the CALCOMP plotter either directly or on tape. Each chart page is 8.5 X 11 in. with one column of flow path. Pages are numbered, and symbols are supplied to chart page-broken logic flow paths. Figure 3 depicts a FLOWGEN chart.

Both flowcharters completely automate the charting of programs. However, most analysts will concede that manually manipulated page topology is generally more acceptable than automated columnized formats, particularly for large, complex subroutines.

AUTOMATED DOCUMENTATION OF PROGRAM INTERNAL COMMUNICATION

Given that a program is composed of a collection of subroutines where the word "subroutine" is a generic term including functions, entry points, block data, and other subelements, certain program internal intersubroutine communication documentation is desirable. Useful cross-reference information would include forward reference, backward reference, and flow hierarchy. Forward references include–

(1) All subroutines referenced by this subroutine
(2) All commons referenced by this subroutine
(3) All global variables defined in common arrays referenced by this subroutine, and
(4) All local variables defined within and referenced only by this subroutine

Reverse references include–

(1) All subroutines that reference this subroutine
(2) All subroutines that reference this common, and
(3) All subroutines that reference this global variable

Flow hierarchy is the cascade of subroutine forward references that presents an overall view of the program flow logic.

Table 1 summarizes those portions of the cross-reference information that are available from the manufacturers' standard software systems: the CDC 6000 compilers and overlay

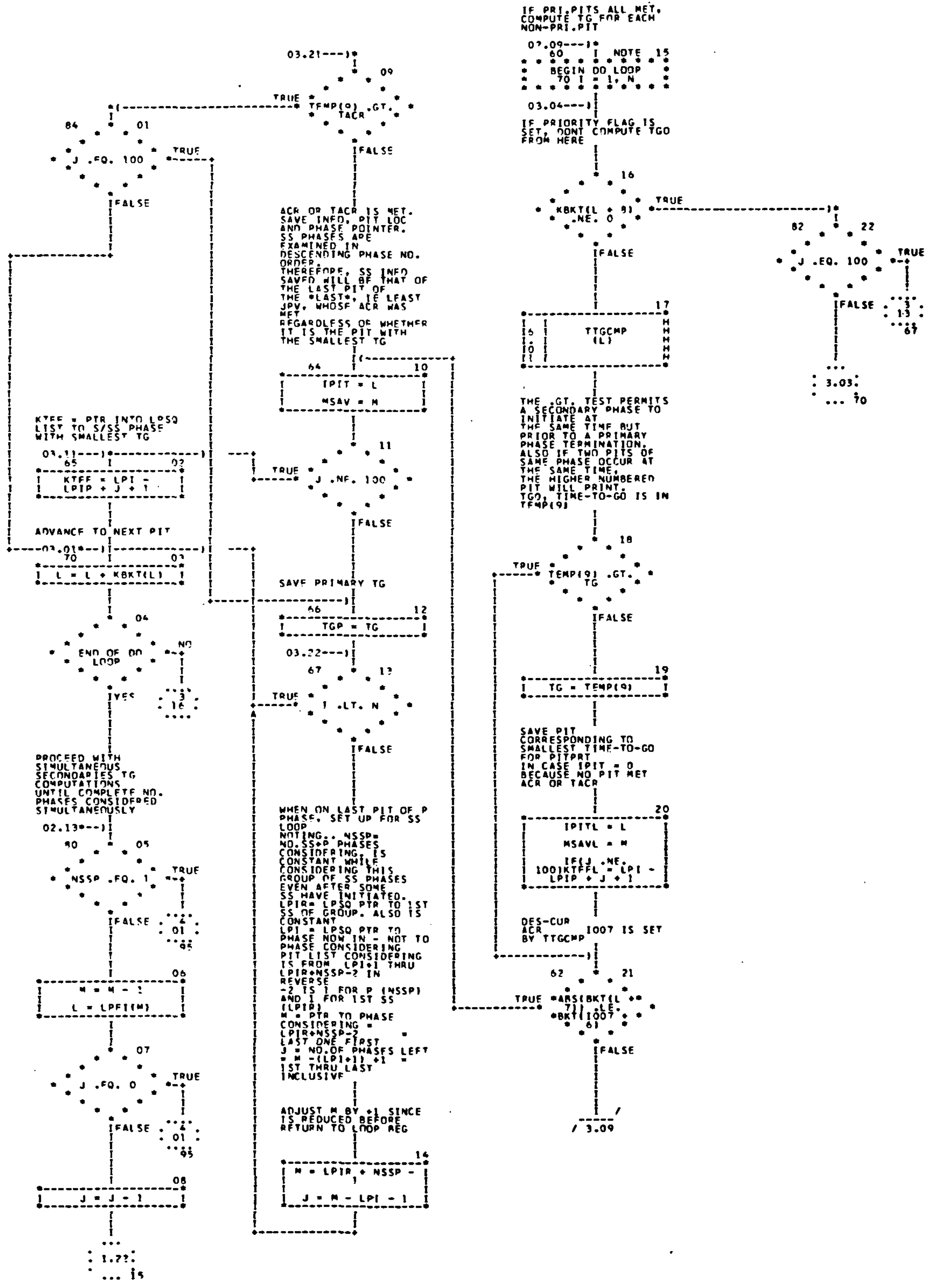
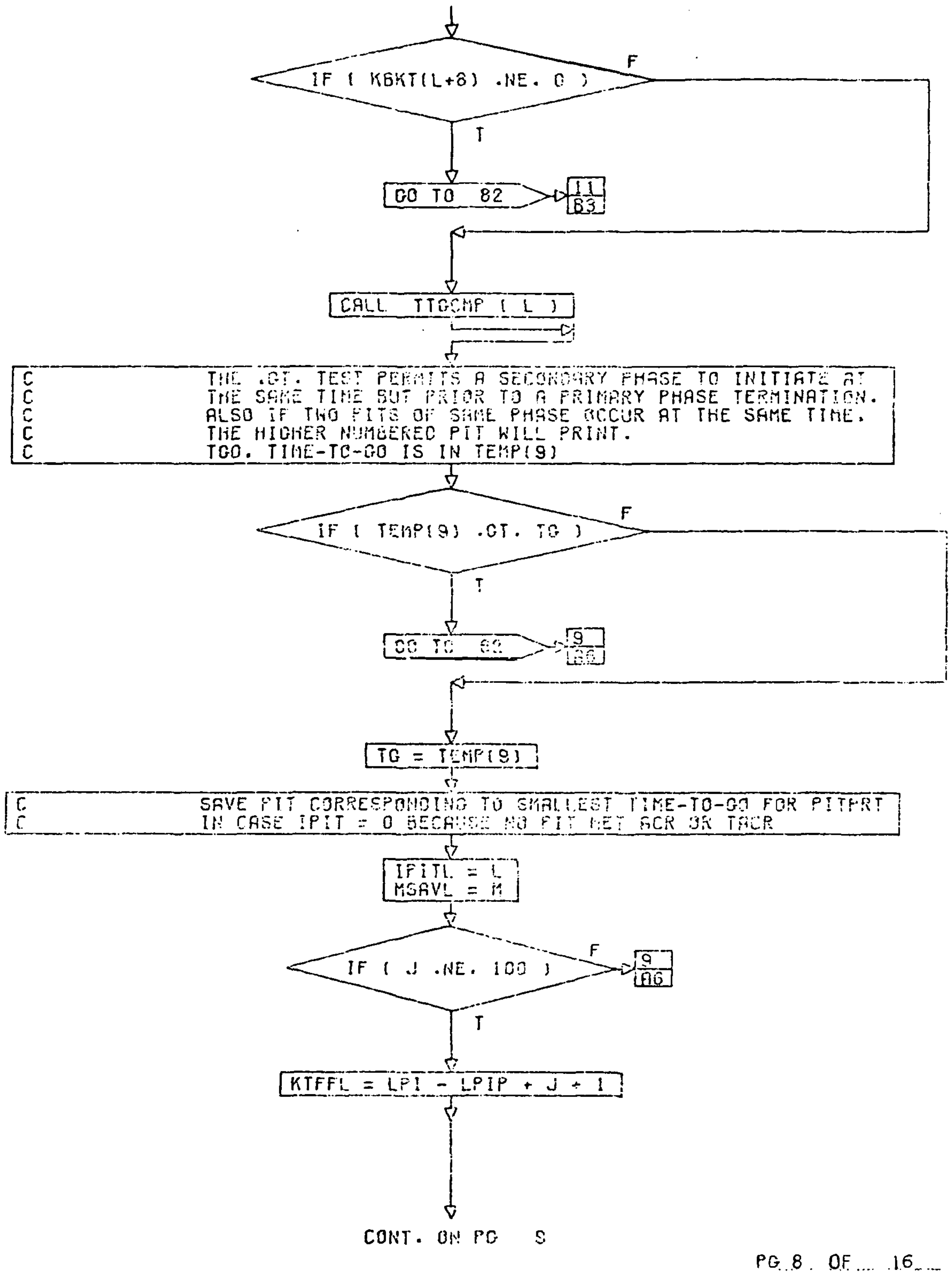

IF PRI.PITS ALL MET, COMPUTE TG FOR EACH NON-PRI.PIT
BEGIN DO LOOP 70 I = 1, N
IF PRIORITY FLAG IS SET, DONT COMPUTE TGO FROM HERE
KBKT(L + 9) .NE. 0
J .EQ. 100
TTGCMP (L)
TEMP(9) .GT. TACR
J .EQ. 100
ACR OR TACR IS MET. SAVE INFO, PIT LOC AND PHASE POINTER. SS PHASES ARE EXAMINED IN DESCENDING PHASE NO. ORDER. THEREFORE, SS INFO SAVED WILL BE THAT OF THE LAST PIT OF THE *LAST*, IE LEAST JPV, WHOSE ACR WAS MET REGARDLESS OF WHETHER IT IS THE PIT WITH THE SMALLEST TG
IPIT = L
MSAV = M
J .NE. 100
SAVE PRIMARY TG
TGP = TG
I .LT. N
WHEN ON LAST PIT OF P PHASE, SET UP FOR SS LOOP NOTING.. NSSP= NO.SS+P PHASES CONSIDERING, IS CONSTANT WHILE CONSIDERING THIS GROUP OF SS PHASES EVEN AFTER SOME SS HAVE INITIATED. LPIR= LPSQ PTR TO 1ST SS OF GROUP. ALSO IS CONSTANT LPI = LPSQ PTR TO PHASE NOW IN - NOT TO PHASE CONSIDERING PIT LIST CONSIDERING IS FROM LPI+1 THRU LPIR+NSSP-2 IN REVERSE -2 IS 1 FOR P (NSSP) AND 1 FOR 1ST SS (LPIR) M = PTR TO PHASE CONSIDERING = LPIR+NSSP-2 LAST ONE FIRST J = NO.OF PHASES LEFT = M -(LPI+1) +1 = 1ST THRU LAST INCLUSIVE
ADJUST M BY +1 SINCE IS REDUCED BEFORE RETURN TO LOOP BEG
M = LPIR + NSSP - 1
J = M - LPI - 1
KTFF = PTR INTO LPSQ LIST TO S/SS PHASE WITH SMALLEST TG
KTFF = LPI - LPIP + J + 1
ADVANCE TO NEXT PIT
L = L + KBKT(L)
END OF DO LOOP
PROCEED WITH SIMULTANEOUS SECONDARIES TG COMPUTATIONS UNTIL COMPLETE NO. PHASES CONSIDERED SIMULTANEOUSLY
NSSP .EQ. 1
M = M - 1
L = LPFI(M)
J .EQ. 0
J = J - 1
THE .GT. TEST PERMITS A SECONDARY PHASE TO INITIATE AT THE SAME TIME BUT PRIOR TO A PRIMARY PHASE TERMINATION. ALSO IF TWO PITS OF SAME PHASE OCCUR AT THE SAME TIME, THE HIGHER NUMBERED PIT WILL PRINT. TGO, TIME-TO-GO IS IN TEMP(9)
TEMP(9) .GT. TG
TG = TEMP(9)
SAVE PIT CORRESPONDING TO SMALLEST TIME-TO-GO FOR PITPRT IN CASE IPIT = 0 BECAUSE NO PIT MET ACR OR TACR
IPITL = L
MSAVL = M
IF(J .NE. 100)KTFFL = LPI - LPIP + J + 1
DES-CUR ACR 1007 IS SET BY TTGCMP
ABS(BKT(L + 71)) .LE. BKT(1007 + 6)
TRUE
FALSE
YES
NO

Figure 2.–Example of AUTOFLOW flowchart.



```
IF ( KBKT(L+8) .NE. 0 )
GO TO 82
CALL TTGCMP ( L )
C                 THE .GT. TEST PERMITS A SECONDARY PHASE TO INITIATE AT
C                 THE SAME TIME BUT PRIOR TO A PRIMARY PHASE TERMINATION.
C                 ALSO IF TWO PITS OF SAME PHASE OCCUR AT THE SAME TIME.
C                 THE HIGHER NUMBERED PIT WILL PRINT.
C                 TGO. TIME-TO-GO IS IN TEMP(9)
IF ( TEMP(9) .GT. TG )
GO TO 86
TG = TEMP(9)
C                 SAVE PIT CORRESPONDING TO SMALLEST TIME-TO-GO FOR PITPRT
C                 IN CASE IPIT = 0 BECAUSE NO PIT MET ACR OR TACR
IPITL = L
MSAVL = M
IF ( J .NE. 100 )
KTFFL = LPI - LPIP + J + 1
```

Figure 3.–Example of FLOWGEN flowchart.

Table 1.—Program Cross-Reference Information Available From Standard System Software

Software system	Forward reference function				Reverse reference function		
	1	2	3	4	1	2	3
Compilers:							
CDC RUN	No[a]	Yes	Yes	Yes	([b])	([b])	([b])
CDC FUN	No	Yes	Yes	Yes	([b])	([b])	([b])
CDC FTN version 3	Yes	Yes	No[c]	No[c]	([b])	([b])	([b])
GE GECOS	Yes	Yes	No	No	([b])	([b])	([b])
IBM 360	Yes	Yes	Yes	Yes	([b])	([b])	([b])
IBM 7094	Yes	Yes	Yes	Yes	([b])	([b])	([b])
IBM 7094 assembler	Yes	Yes	No	Yes	([b])	([b])	([b])
Loaders:							
CDC loader map	No	Yes	No[b]	No[b]	Yes[d]	No	No
GE loader map	Yes	Yes	No[b]	No[b]	No	No	No
IBM 360 linkage editor	No[e]	No[e]	No[b]	No[b]	No	No	No
IBM 7094 load map	No	Yes	No[b]	No[b]	No	No	No
IBM 7094 logic map	Yes	Yes	No[b]	No[b]	Yes	Yes	No
NASTRAN linkage editor	Yes	No	No[b]	No[b]	Yes	No	No

[a]TRW has modified the CDC RUN compiler to output the referenced subroutines.

[b]Not applicable for the compilers and perhaps desirable from the loader maps only as an option.

[c]An option causes the local variables to be printed and the locations of all references to a common array to be listed; it does not print the names of the global variables.

[d]The subroutine reverse references are available only within an individual program overlay.

[e]An option generates the name of the referenced subroutine and/or common along with the locations, not names, of the referencing subroutine.

loader; the IBM 360/50/65/85 levels G and H compilers and linkage editor; the IBM 7094 IBSYS compiler, assembler, and IBLDR loader; and the NASTRAN loader, which operates like the IBM 360 linkage editor on the CDC 6000 computer.

As can be seen, five of the seven compilers mentioned generate the subroutines referenced by a given subroutine. All the compilers give the referenced commons. Four compilers list the global variables, and five list the local variables referenced by the subroutine. Three of the six loaders give forward subroutine references, and four give the commons referenced by a subroutine. No loader names a variable. This would be desirable from the loader map as an option, but the variable names are probably not immediately available.

The reverse references for subroutines are for all practical purposes missing from four of the loaders, and only one generates all references to a given common. Nowhere is information available to yield all references to a global variable, which is highly desirable in validating and maintaining a program.

No standard loader generates a complete expansion of the forward subroutine references, which also is desirable in using a program.

To surmount these deficiencies and to provide what is deemed useful documentation, TRW has developed several programs. Figures 4 and 5 represent their outputs.

Symbol De. Name

Name	LKZ	DLK	DTA	CLT	UTL	OPT	ERR	INT	IPT	CRD	PRT	PRM	PRN	PRO	PRP	PRQ	PRR	BSF	COM	FST	RSO	NRS	CHK	TRT	TTI
00ERR					U		B																		
06ERRI							B																		
06ERRA							B																		
06ERR					U		D																		
08ERR					U		D																		
09ERRC							B																		
09ERRN							B																		
09ERRR							B																		
09ERR					U		B																		
09ERRS							B																		
0BOFF			D																	U					
0BTYP																				B					
0L1			D					U																	
0L2			D																						
0L3			D													U									
0MP			B		U					U		U						U	U	U					
0MR			B									U							U	U					
0MY			B									U							U	U					
OPTION			D		U		U	U		U	U	U	U	U		U				U	U	U	U	U	
ORDER			D					U												U				U	
106RK												B			U										
106RKZ												B													
10AZZA									B																
10AZZB									B																
10AZZC									B																
10AZZ									B																
10AZZX									B																
10ERR					U		D																		
11ERR							D																		
12X4									B																
16ERR					U		D																		
17ERR					U		D																		
19ERR					U		D																		
1AAZZB									B																
1AAZZ									B																
1ACGC													D			U								U	
1ACGCZ													B												
1ACTD1					B																				
1ACTD2					B																				

Figure 4.–Example of global variable cross-reference, IBM 7094 program.

```
SYMBOL        SUBROUTINE

A             NVRS3D    TVGNL     TVGSC
AA            NVRS3D
AAP           AANGT     APRIT     BCNFT
AAPM          AANGT
AAT           AANGT     APRIT     BCNFT
AAY           AANGT     APRIT     BCNFT
AAYM          AANGT
ACSOR         CREAD     INPROC
ACTNMR        CRFAD     INPROC
AERC          AATMP     AFFDR     AIRVEL    APRIT     DCNFT     BLKDTA    CREAD     ARBIT
              MDRE      MDRAG     SGFFL
              RTAMP
AHT           TM1PS     T1AFF     TM1NS     TNAFF     T2AFF     T3AFF     TVGCD     TVGMC
              TVGSC
AIRSHP        CREAD     INPROC
ALATL         ASTART    CREAD     INPROC    INPRUC
ALMCOF        TVGMC     TVGNL
ALTGT         INPROC
AMUL          ASTART
AMUT          BIMAZ     TM1PS     TM1NS     T3AFF     TCCTD     TVGMC
AMXLOO        MCDER
APAF          AATMF     AATMP     ADVIC     AFFDR     AIRVEL    APFDR     APRIT     ASPAC
              ATGOE     BCNFT     BLKDTA    MDRAG     MTHWF     SGFFL
APDPT         TVGNL
APDR          TVGNL
APSI          AIRVEL    APFDR     ATMOS     MTHWF
ASTRH1        BSTRS3    SGBST
ASTRH2        BSTRS3    SGBST
ATEMP         AIRVEL    ATMOS     NVRS3D
AZCORR        ASTART    INPROC
AZL           APRIT     ARBOR     ASTART    AZEST     CREAD     ARBIT     T3AFF
AZL1          T3AFF
AZMUT         ANIP      BIMAZ     TCCTD
AZPTS         CREAD     INPROC
AZSECT        CREAD     INPROC
B             NVRS3D    TVGNL     TVGSC
BB            NVRS3D
BBARCO        TVGCD     TVGMC     TVGNL     TVGSC
BETA          AAUXF     APRIT     ATCRS     ATGOE     MDRAG     TM1PS     TM1NS     TMLIN
BETI          AAUXF     APRIT
BIGEST        ATGOE
BLANK         INPROC
BURST         CREAD     INPROC
BURSTA        CREAD     INPROC
BURSTB        CREAD     INPROC
C             TVGNL     TVGSC
CALM          ACDER     ANAVH     ANIP      BLKDTA    EGRAV
CASE          INPROC    ?RGFNC
CD            AATMP     AFFDR     AIRVEL    APRIT     MDRAG     SGFFL
CDDELT        MDRAG
CDMULT        AATMP     AFFDR     MDRAG     SGFFL
CDPR1         MDRAG
CDPR2         MDRAG
CDPR3         MDRAG
CDREF         MDRAG
CGLATL        AZEST     INPROC
CGM           AD2CG     SGBST
CGOFFS        AD2CG     APFDR
CGOFFT        AD2CG     APFDR
```

Figure 5.—Example of global variable cross-reference, IBM 360 program.

Figure 4 presents the output of the symbol reference program, which lists every global and local variable along with the name of the deck that defines (D), uses (U), or both defines and uses it (B). Deck here may be a collection of subroutines. Written in IBM 7094 assembly language and operating under the IBSYS system, this program documents an IBM 7094 assembly language program from its output list tape.

Figure 5 presents the output of a similar symbol reference program, which lists the global variables with every subroutine reference. This program is written in IBM 360

FORTRAN and uses as inputs the common and equivalence statements from FORTRAN subroutine source cards.

This variable reference information may also be generated for other IBM 7094 machine language programs that operate in the TRW SCAT system. This additional capability for the ADS program is currently under review for the CDC 6000; it is designed to document FORTRAN IV programs and to accept CDC list tapes and/or source cards as inputs.

Figure 6 presents the flow hierarchy of a program that is the output of AFLOP; it expands the subroutine references until it exhausts the calls or reaches an undefined external. It currently operates on the IBM 360 and the CDC 6000 computers. AFLOP accepts input cards that spell out the names of the referencing and referenced subroutines. An option permits subroutine expansion at each encounter or line number reference to the first expansion. TRW intends to incorporate AFLOP as an option of the CDC 6000 NASTRAN linkage editor and to make it available for the CDC 6000 overlay loader; in these systems it acquires its inputs from the loader tables.

ADMINISTRATIVE TERMINAL SYSTEM

Formerly, certain documents, such as programmer's handbooks and users guides, have been prepared manually. A revision usually meant considerable retyping and proofreading, both of which consumed time and could introduce errors. Several text editors have appeared on the market; a good one is the ATS developed by IBM under the acronym DATATEXT. TRW currently purchases time to use this system and is contemplating installing the program in-house.

The system uses an IBM 360 computer with appropriate storage devices and high-speed printers at the central site. Telephone lines connect the computer with the remote stations. The terminal may be an IBM 2741 or a DATEL 30, and either may be hard wired or connected to a standard telephone set with a data coupler. The keyboard resembles an IBM Selectric typewriter and may be used as such when disconnected from the computer. The operator-secretary enters a document by instructing ATS with margin placements, tab settings, and formats; the text is typed and the system is requested to file it in permanent storage. ATS assigns line numbers for subsequent editing.

Features of the system include indented and blocked paragraphs, page width and depth control, page headings and footings with automatic centering and numbering, line and page skip, margin justification, and table and chart special formats. Lines may be kept together; for example, in a table that should not be split across pages. Form letters may be prepared, and, with the stop code, one may request the printing to halt temporarily for the insertion of particular data.

The print options include printing some portion or the whole document with or without line numbers, with or without justification, at the terminal or on the high-speed printer. ATS displays its agility in the editing capabilities. Corrections reference the line number and any word within the line. One may remove or replace a word, a phrase, or a line, add to or remove lines, and physically move lines or paragraphs. The edited document as well as the original version may be retained in storage.

```
 1MISCNT
   2    ADDPNT
   3            ADDRES
   4            MRELSE
   5                    CFC
   6            OUTPTC
   7                    GETBA
   8                    SYSTEM
   9                    ABNORML
  10                    OPEN.
  11                            ADVIN.
  12                    STO.
  13                    DAT.
  14                    POSFIL.
  15    CMPINC
  16            PHSCNT
  17                    PSREAD
  18                            REWINK
  19                                    SYSTEM
  20                                    ABNORML
  21                                    GETBA
  22                                    CIOI.
  23                                    ADVIN.
  24                            OUTPTC    6
  25                            KILRUN
  26                                    NAMLOC
  27                                    OUTPTC    6
  28                                    LINK
  29                            INPUTB
  30                                    GETBA
  31                                    SYSTEM
  32                                    ABNORML
  33                                    CIOI.
  34                                    STO.
  35                                    OPEN.     10
  36                                    MVADS.    *
  37                            PT
  38                                    LOCF
  39                                    OUTPTC    6
  40                            MVSPRD
  41                    STAGCN


  42                            PRESET
  43                                    PHSVAN
  44                            UNPKIO
  45                            OUTPTC    6
  46                            IPT       *
  47                            KILRUN    25
  48                            PITSTS
  49                                    DATSTO
  50                                            OUTPTC    6
  51                                    UPITMD
  52                                    VARSET
  53                                    SINOUT
  54                                            ACCUFR
  55                                                    SYSTEM
  56                                                    ABNORML
  57                                            OUTSYS
  58                                                    OUTPTC    6
  59                                                    COPYDA
  60                                                    OCHDYC
  61                                                            EPHEMX
  62                                                                    INTR2
  63                                                                            FILOPN
  64                                                                                    PT        37
  65                                                                                    KILRUN    25
  66                                                                                    OUTPTC    6
  67                                                                                    PT        37
  68                                                                            IOINT
  69                                                                                    SYSTEM
  70                                                                            DBLE    *
  71                                                                            SNGL    *
  72                                                                            BACKSP
  73                                                                                    SYSTEM
  74                                                                                    ABNORML
  75                                                                                    GETBA
  76                                                                                    CIOI.
  77                                                                                    BKSPRU.   *
  78                                                                                    FIZPAK.
  79                                                                                    ADVIN.
  80                                                                                    FIZPA.
  81                                                                                    POSRU.    *
  82                                                                            REWINK  18
```

Figure 6.–Example of AFLOP.

Some inherent disadvantages are the lack of superscript and subscript notation and the omission of special characters such as the Greek alphabet. The preparation of equations manuals can be only partially automated; the text could be maintained using this editor with space allowed for the manual entering of the mathematical equations or diagrams.

The ATS system provides various administrative facilities such as a log of the documents stored; the date, name, and size of each; and the total number of documents in storage. Complete or partial storage reports may be requested. Each document may contain a password that prevents anyone who does not know the password from accessing the document. The password may be changed as often as desired to achieve some level of security.

The MATS manuals are partially maintained via ATS, and it is intended that all future documentation be implemented with this system.

Brief mention should be made of other text-editing programs. One is the TRW General Trajectory Documentor (GTDOC) program. It accepts a file of prebuilt text from tape or cards, a card deck of text modifications, and a data set of trajectory parameters either from tape or cards. The output is a standard-form document with the trajectory data positioned properly in the text. GTDOC automates the preparation of trajectory-oriented publications. It operates on the IBM 7094.

Although the TRW Timeshare System Editor is designed primarily to aid in preparing executable programs, it incorporates commands useful in constructing other types of data files.

COMPUTER-AIDED PROGRAM DEVELOPMENT PROPOSAL

Occasionally, it is advantageous to analyze the procedures normally pursued in the development of a program. In addition to the normal preliminary functions of defining and specifying the usual program performance criteria, the steps involved are—

(1) Creating a flowchart
(2) Defining the flowchart
(3) Generating the program code
(4) Analyzing the coding errors
(5) Analyzing the design inadequacies
(6) Iterating (2) through (5) until complete
(7) Documenting the subroutine(s) by redrawing the flowchart(s)
(8) Developing the program further by iterating (2) through (7)

Considerable time is expended in generating a flowchart from which the programmer/analyst prepares the program, as any engineer or analyst can readily attest. Refining this flowchart to introduce even one new equation requires providing the right space at the right place (foresight), erasing and shifting the symbols with contents (copy errors), and redrawing and shifting (copy errors).

The code is then revised to match the flowchart, which often results in inefficiency in the code and a generally disorganized arrangement in both the sequence of operations and format of the subroutine. Eventually, a new flowchart is necessary. Program evolution consists of flowchart, code, analysis, flowchart.

CAPD proposes a system wherein the code and the final flowchart no longer appear as steps in program development. This technology uses a graphics display console with character, line, and vector capabilities. The IBM 2250 and the CDC Digigraphics consoles are potential candidates. Time sharing is desirable for economic purposes only.

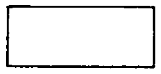
The initial flowchart is created with aid from the computer, which provides the flowchart symbols and automated spacing. Modifications are achieved with maximum ease and reliability with the CAPD graphics program. The analysis proceeds at an accelerated pace because CAPD permits the analyst to concentrate on the problem, aids in diagnosing the flow diagram, and supplies definitive information on request or on repeated error occurrences.

CAPD Translator

The translator takes input from the graphics flow diagram in terms of arithmetic and logical expressions enclosed within the flowchart symbols. It transcribes this flowchart into source language appropriate for compiler input by translating, for example, rectangles into arithmetic statements, hexagons into CALLs, and triangles and diamonds into IF statements.

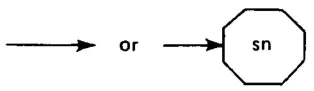
CAPD Conventions

The conventions follow FORTRAN closely and adopt common flowchart symbol graphics representations:



Arithmetic statements: rectangle. The operators +, -, *, /, **, or ↑ and the field delimiters =, ,, () carry FORTRAN definitions.

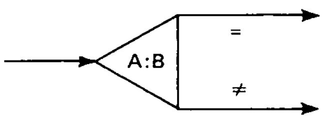
Control statements



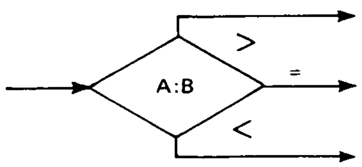
Unconditional transfer: directed arrow to a statement number (sn) enclosed in an octagon

Conditional transfer

The colon is introduced for comparison of expressions followed by directed arrows.

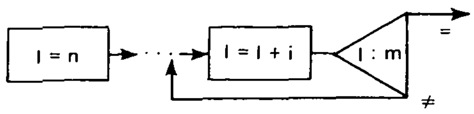


Equality: triangle

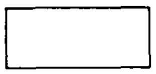


Inequality: diamond

A and B in the diagrams may each be arithmetic expressions. These representations encompass the GO TO and the IF statements.



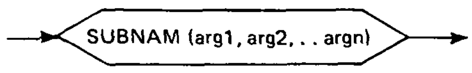
The DO statement is easily represented by a combination of arithmetic and conditional transfer expressions indicating a loop on I from n to m by i.



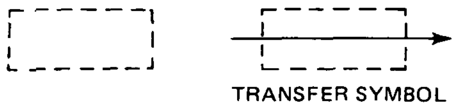
Input/output (I/O) statements: rectangles. Format statements are written and spaced exactly as they are to appear on the printed page, ###.# or .#######; e.g.,

TIME ###.### WEIGHT .######## E##

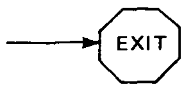
Declarative statements: no flowchart symbol. These follow the FORTRAN specifications except that the words are prebuilt and the analyst may point to EQUIVALENCE rather than spell it. END indicates completion of the subroutine flowchart.



Subroutine execution: hexagon. The CALL or Return Jump is internally generated.



Comments: perforated rectangles. These statements may be placed anywhere on the flowchart such that they do not interfere with any real statement except the transfer symbol.



Exit: octagon. This is placed on the flowchart where a RETURN statement is to be simulated.

CAPD Graphics

The graphics program automates the flowcharting process by providing space for insertion of new statements and by collapsing the flowchart or extending the arrows as erasures are made.

Today's technology provides at least three methods for sketching and writing on a display device.

(1) Typewriter–alphanumeric characters are immediately available; flowchart symbols could be defined as–

[ ] = rectangle	[= beginning	] = end
<> = diamond	< = beginning	> = end

3 of 3

(2) Menu–characters and symbols are presented on the display and they are initialized by pointing to them with a light pen and indicating a location; e.g., pointing at a rectangle causes a nominal sized rectangle to appear on the display face where indicated, and statements are written inside it.

(3) Character recognition–the analyst draws a rectangle that is "recognized" by two nearly horizontal and two nearly vertical lines; the display presents an internally blocked and sized flowchart symbol or alphanumeric character at the current cursor location.

In either of the latter two methods the flowchart symbol size is modified by "pulling on its handle." A similar method may be devised for moving the symbol with its expressions intact to a different location on the display.

The method of character recognition is probably best suited to the normal analyst. Flowchart symbol recognition combined with typewriter alphanumeric input may be most efficient for keypunch operator use. A menu of flowchart symbols combined with typewriter alphanumerics may seem simplest to implement.

CAPD I/O

In program design it must be possible to initiate a flowchart and at a later date reintroduce it to the computer for additional development.

Inputs to CAPD consist of a graphics flowchart created by the analyst at the console and a previously executed flowchart as output by CAPD. Conceivably, an optical scanner device could be programmed to re-create the graphic flowchart identical to the original; otherwise, an alternate form of input would be made available.

Outputs from CAPD consist of hard copy of the graphic flowchart (optional computer output microfilm); source language for the appropriate compiler (possibly an option of punching the source language on cards); and, in the event that an optical scanner is unfeasible for inputting a previously generated flowchart, a representation of the flowchart on cards, tape, or other media.

The only restriction applicable to I/O is that CAPD generates certain output such that it may become input to itself.

CAPD Diagnostics

A flowchart convention may exist to prohibit the analyst from leaving an unfilled flowchart symbol; thus, if he does not know precisely what the symbol is to contain, he writes a question mark (?) and is allowed to proceed. This permits CAPD to examine each flowchart symbol and report omissions or errors as they occur. Validation of the calling sequence arguments with library subroutines is a potential diagnostic. When CAPD receives the END signal, it questions the analyst concerning the unfilled flowchart symbols, the undefined transfer points, and formats. The diagnostic output provides the analyst with the correct format of any symbol or expression and automatically displays itself if he commits an error repeatedly.

Successful compilation is almost assured; successful execution depends on the response to the diagnostics. The analyst may elect to ignore or leave incomplete portions of the subroutines; CAPD will not inhibit use of the compiler if the user specifically requests to proceed.

Programming Reliability

The diagnostic remarks assist in immediate error recovery. Pictorial representations are considerably less error prone than word images. Modifying a flowchart, where such is possible without copying it, is nearly always performed accurately, whereas generating the code requires particular attention to the format, punctuation, and logical assumptions of the language.

Quick Response and Rapid Reaction

The calendar time required to design or modify a program is drastically reduced by automated regeneration of flowcharts as refinements or alterations are introduced and automated translation of flow diagrams into source language. The implementation of a new or revised set of guidance equations, for example, would take relatively little time compared to today's normal turnaround time. This technology provides real-time systems with fast and accurate response.

Documentation

CAPD reverses the entire documentation procedure. Contracts normally oblige the analyst to document the program. With CAPD the modus operandi is to "program the document." Flowcharting the subroutine is no longer necessary, and the remaining documentation would be formatted as described in ADS to permit complete automation. Thus, the portions of documentation that are always tedious and laborious to produce are bypassed. A very important result is that the readability and reliability of the program are greatly enhanced.

CAPD Recommendations

The implementation of CAPD should be seriously considered by developers of computer software. Not only would the state of the art take a major stride forward, but considerable cost effectiveness would ensue from the diminished time required to create, update, maintain, and document a program.

RECOMMENDATIONS AND CONCLUSIONS

The computing industry has already accrued considerable benefit from the ANSI standards for the FORTRAN language; programs that adhere to these specifications transfer readily to another computer system. There should be similar standardization for loaders. As yet ANSI either has not addressed this problem or has not felt sufficiently fortified to assert itself to this technically feasible but politically delicate problem.

NASA, an important customer of the computing industry, now asks whether program documentation in a broad general sense can be automated. The response is assuredly positive if standardized specifications can be outlined and accepted. Of course, each software developer has and can continue to automate his program documentation individually, but from the buyer's viewpoint cost effectiveness is not necessarily achieved. Certain standardizations are considered in the following recommendations.

Program Specifications

The specifications for programs contained in requests for proposals occasionally present problems to the responder by putting him in doubt as to the relative complexity, generality, and capability of the product desired. This author recommends the topic as the subject of a future symposium.

Manufacturer Software

Much internal program information that is immediately available from the compilers and loaders is lost simply because it is not printed. As discussed in the section entitled "Automated Documentation of Program Internal Communication," forward and reverse subroutine and common reference information is invaluable documentation. It is recommended that the manufacturer of software systems provide options to retrieve the information so that all applicable items in table 1 may be marked affirmatively.

Program Development

The CAPD system discussed in the CAPD proposal should be seriously considered for the best use of engineer and analyst time and to reduce a major portion of the effort expended in documenting a program.

Program Documentation

Many items of subroutine level documentation are considered standard; such as the name, title, author, abstract, calling sequence, restrictions, and variables usage. The format of these information files remains to be defined; this author recommends coded comments.

It may be presumptuous to anticipate that the industry could agree on those items to be retained in the program listing, in the flowchart, and in the documentation. Therefore, the code should contain options that permit the commentary to appear on the respective documents as requested, thus avoiding duplication without sacrificing completeness. The coded comment cards defined for the ADS program discussed represent a step in the correct direction, but additional refinements along with modifications to the current compilers, flowcharters, and documentation programs must be specified, standardized, and implemented.

Manuals Preparation

Input, output, and deck setups are fairly common subject titles in most users manuals. Having directed the development of a large, complex, general-purpose program, the author is aware of some inadequacies of this type of manual. The customer requires problem-oriented information.

It does seem clear that manuals should be generated with automated text editors as mentioned earlier in the paper. An economy of operation is realized, particularly for high-usage dynamically evolving programs, when the documentation can easily and accurately be created and updated.

DISCUSSION

MEMBER OF THE AUDIENCE: I would like to know to what extent the information you have here is available to the general public. In other words, is it proprietary?

LANZANO: Yes, the programs themselves are proprietary. They are for sale.

MEMBER OF THE AUDIENCE: You indicated that this was tied directly to your trajectory determination program. Is there anything within that program that restricts its use?

LANZANO: I did not mean to imply that it is tied directly to this program. It was developed to support this program. It would support any program that follows the set of standards that I defined. We also have other programs that will handle all machine-language programs in a somewhat similar manner; therefore, they are strictly supportive programs.

MEMBER OF THE AUDIENCE: This is quite an involved system. How many years has this been in process?

LANZANO: I think it has actually been in process for 3 or 4 years. They are not difficult programs to write. Most of them have been written as kind of off-the-cuff things. The ADS program is probably the more difficult because it mimics the compiler. In direct answer to your question, it has evolved over a period of years, but the level of effort in producing this is not particularly high.

MEMBER OF THE AUDIENCE: In this area of standardized loads, do you think an extension in the FORTRAN standard that identifies the requirements for overlays and segmentation would be useful?

LANZANO: I would like to see some standards defined for overlays and segmentation. Whether such standards would be useful would depend. I have found the IBM 360 linkage editor to be quite versatile. I think it is probably one of the best in the field right now. The Univac 1108 looks very good to me, although I have not actually used it.

MEMBER OF THE AUDIENCE: The only other question I have is in the area of the technical editor. Have you had any occasion to wish that the technical editor would do a quick index for you on the document, so you could actually go through and take your document and see how well you have used the same phraseology so that it becomes easier for the reader?

LANZANO: This one does not, but there is one called QED developed by Time-Share that will look for phrases.

MEMBER OF THE AUDIENCE: I mean a quick index of the whole document from the form that you put it in.

LANZANO: I believe the one that Dr. Rich discussed actually put out a table of contents at the end.