N73-19212

MAKING AUTOMATED COMPUTER PROGRAM DOCUMENTATION A FEATURE OF TOTAL SYSTEM DESIGN

Allan W. Wolf
System Development Corp.

The "paper-mill" character of large-scale computer software systems is a condition that is all too familiar to anyone involved in the computer programming business. Program manuals, design specifications, administrative reports, system descriptions, and user's manuals are just a few of the kinds of documents necessary to support a big software system. These documents, besides being complex, abundant, and subject to change, are frequently afterthoughts to the systems they support rather than part and parcel of the system design. This factor, when coupled with deadline and money pressures, unfortunately leads to another all too familiar condition—inadequate documentation.

Program documents are too often fraught with errors, out of date, poorly written, and sometimes nonexistent in whole or in part. This condition need not exist, however. Data stored on the printed page should be accurate, accessible, and helpful to the user, and it can be if a systems approach and existing computer technology are employed. This paper describes how many of these typical system documentation problems were overcome in a large and dynamic software project.

The project that will be discussed is the U.S. Air Force Satellite Control Facility (AFSCF) orbital prediction and command system. It is both large and dynamic and consists of about 2.5 million machine instructions in some 900 programs, over one-half of which are being modified during a typical 6-month period. The documentation supporting this system amounts to some 65000 pages, and an average of 5500 new and modified pages are published each month.

At one time there were numerous system problems that could be attributed to a lack of quality in the software documentation. More than 10 subcontractors produce the satellite computer program subsystems for AFSCF operations. It is common to have several of these agencies simultaneously produce programs that must interact (or interface) with each other and also with existing software. Before the development of the current system, this procedure led to problems in computer storage sharing, program calling sequence interpretation, interface data design, and other areas. In addition, there were all the usual problems associated with poor documentation.

(1) Users did not know how to call or use existing programs. Required input parameters and data could not be determined, and, in some cases, it could not even be determined whether routines existed to perform a particular function. Expensive computer time was wasted experimenting, or essentially duplicate routines were produced because it was felt to be cheaper than the process of decoding existing ones.

(2) Analysts could not determine whether existing routines were adequate for certain applications. References to the mathematical bases for programs were lost, or original work was never documented.
(3) Maintenance and development programmers spent much time and money attempting to modify programs. Great savings could have been realized if there had been adequate standards and conventions or system documents to provide some insight into what had already been done.

As the AFSCF system grew larger and more complex, and as these documentation problems manifested themselves in various ways, it became obvious that they would have to be solved, or the system would become totally chaotic. System Development Corp. (SDC) was given the task of designing and developing a new software system to support AFSCF. There were several design goals for the new system, but the one of primary interest for this paper was the alleviation of the types of problems emanating from inadequate documentation.

To fulfill this and the other design goals, a total systems approach was used. For documentation, this means that each system component was designed with the documentation problem in mind, instead of a procedure that designs the system and considers documentation as an appendage to be developed after and around the basic design. Naturally, there were compromises because of conflicts among the several goals, but the final result was a system that directly incorporated features in the basic design that overcame many of the previous documentation problems. The systems approach encompassed such items as—

(1) Configuration management (a closely monitored software management scheme that guides products through the various design, development, and acceptance milestones)
(2) Standards and conventions (guidelines, restrictions, and quality assurance measures covering many of the program design and development activities; application handled by configuration management)
(3) Collection of program information into central data banks to which all system components will have access, permitting easy and accurate documentation)
(4) Interaction among executive, compiler, central data banks, and configuration management (to provide a check and balance system that will prevent errors)
(5) Automatic documentation (to provide timely and accurate documents)

Figure 1, which will be discussed in detail in the section entitled "Configuration Management," shows the flow of new products through the milestones in the AFSCF system. This illustrates the interaction among some of the items just mentioned.

This paper shows how the system approach guarantees the accuracy of various portions of the documentation, provides the user with a total picture of the system, provides calling sequence and internal information on the various system programs, and, in general, eliminates many of the problems that typically arise from poor documentation. Specifically, the following elements will be discussed:

(1) The data-base definition, or common pool of information (COMPOOL), which is a data base in itself, contains descriptions for every program and piece of interface data (between programs) in the system. It plays a major role in the generation of automated documentation, the simplification of program maintenance, and the minimization of program development costs.

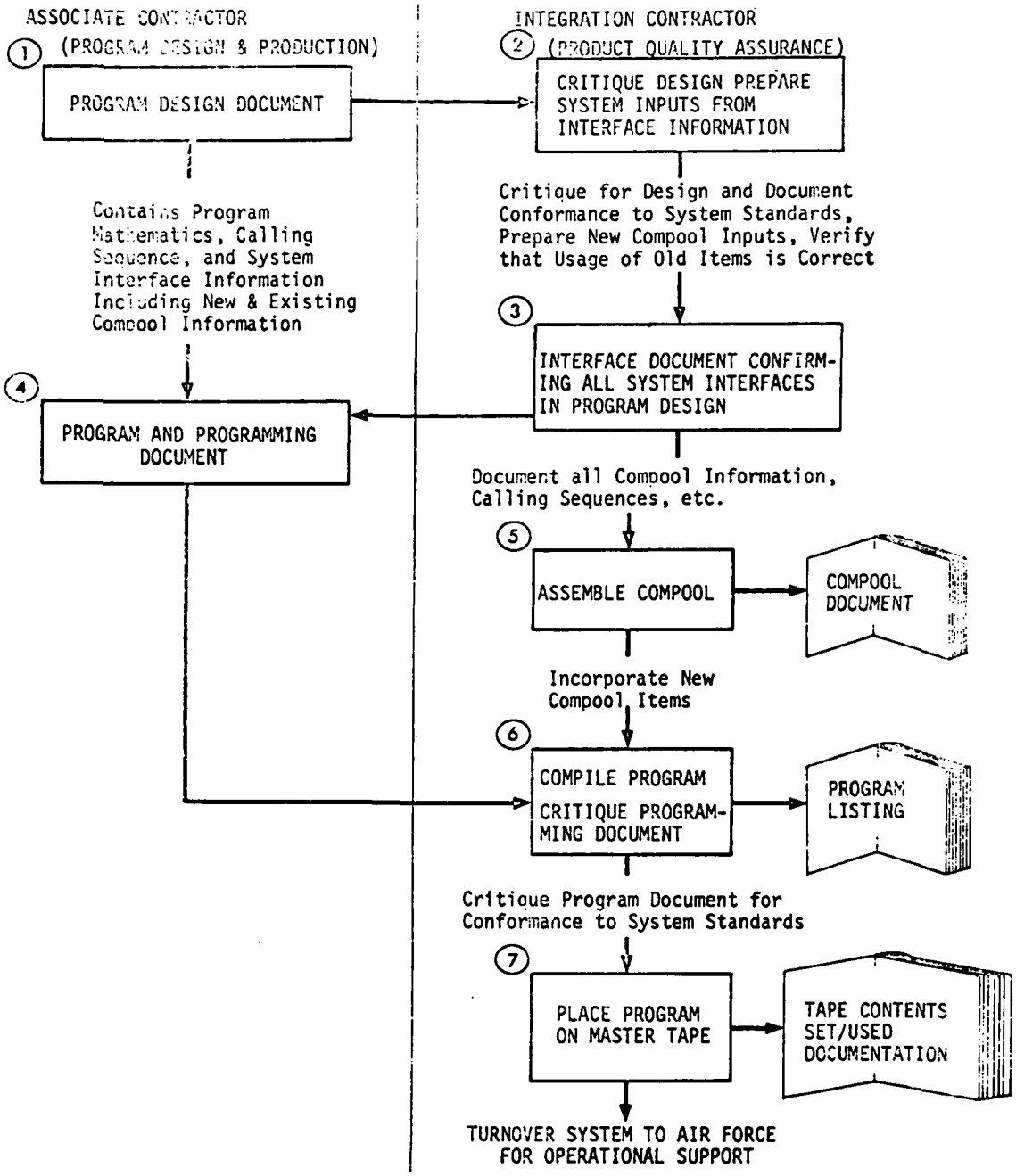

Figure 1.–Product flow through AFSCF configuration management milestones.

(2) The configuration management scheme, although not specifically a part of the software system, is so basic for insuring product quality that any discussion of documentation would be incomplete without a description of it. How production and quality assurance functions check and balance each other, and how new products flow through this controlled scheme will be discussed.

(3) The computer-produced documents, when combined with manually prepared portions of the design and programming documents, provide high-quality, total-system documentation. When and how these documents are generated and the information that they contain will be discussed.

DATA-BASE DEFINITION

There are five basic elements in the AFSCF orbital prediction and command system that will be referenced throughout this paper. These are COMPOOL, the library tape, the master tape, configuration management, and the system programs. Of these, COMPOOL is the most fundamental, for it provides, in one centralized location, the basic definitions, references, and formats used by all the programs in the system.

Specifically, COMPOOL consists of names and calling sequence descriptions for all AFSCF system programs, along with descriptions of all system intercommunication data. Initially, COMPOOL takes the form of a punched card deck containing all the necessary descriptive information in the prescribed format. A special COMPOOL assembly program then processes these cards and compiles the data into tables that are an efficient input for the system compiler. During the COMPOOL assembly, a tape is produced that contains all the information on the input cards. This tape forms the basis for later COMPOOL updates, and it also serves as an input to a program that generates all the COMPOOL documentation. Data in the COMPOOL are organized according to the following hierarchy: blocks, tables (or arrays), and items, blocks being the gross data sets and tables and items being subsets of the blocks. As has been mentioned, COMPOOL also contains program descriptions and calling sequences.

Figure 2 is an example of the program calling sequence that is a part of the COMPOOL output. The explanation for labels *a* to *e* are as follows:

a: The notation SUBR indicates that the data that follow are for a subroutine (computer program). All programs are identified by SUBR.

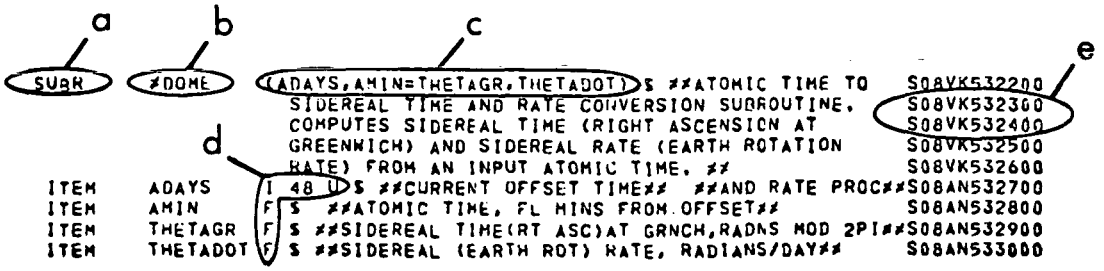

Figure 2.–Program calling sequence (excerpt from COMPOOL document).

b: DOME represents the name of the program (the ≠ symbol precedes all COMPOOL names).

c: These are input and output parameters. All parameters to the right of the ≠ sign are output by the program.

d: These are format definitions for the input and output parameters. The symbols I, 48, U, and F designate such information as floating, integer, signed, and the number of bits occupied by this item. The general data structures are defined in a system standards and conventions manual.

e: These are sequence numbers that permit updating the data definitions on tape without having to work with the entire card deck.

The definitions, enclosed within the ≠ ≠ symbols, are not required by the software system but are supplied for almost all entries. This is, of course, one of the capabilities that makes the COMPOOL document so valuable to users of the system.

Figure 3 shows a sample of the program interface data. The explanations for labels *a* and *b* are as follows:

a: The notation BLK indicates a data block, V indicates the type of block (variable length), and R means that the data are automatically retrieved and stored at set intervals during a run to allow for restarts and subsequent runs. All entries are either SUBR or BLK entries, although the type of block and the retrieval information may vary. The balance of the information describes the data and format of the BLK in a form that is similar to the calling-sequence item descriptions in figure 2. All of the format information is defined in the system standards and conventions manuals.

b: The symbols C and I to the right of the sequence numbers indicate changes to and insertions in, respectively, the previous version of COMPOOL, which was used to produce this one.

```
VBLKR  ≠OAJ      R 48 RV $ ≠≠ORBIT ADJUST TABLE PLUS POINTERS FOR    S08VK199100
a                  SUBROUTINE, ≠DOAR BLOCK, CONTAINS ORBIT ADJUST   S08VK199200
                   INFORMATION FOR A VEHICLE. ≠≠                    S08VK199300
  TABLE  ≠OAJTBL   V 64 P 10 $                                      S08AD199400
   BEGIN                                                            S08AN199500
    ITEM  ≠OAJNO   I 8 U 0 0 $    ≠≠OAJ NUMBER≠≠                    S08VE199600
    ITEM  ≠ASTAB   S 2 V(INERTIAL) V(GEODETIC) V(GEOMAG) 0 8 $      S08AN199700
                       ≠≠STABILIZATION TYPE≠≠                       S08AR199800
    ITEM  ≠ASTAT   S 2 V(PLANNED) V(LOADED) V(EXECUTED) 0 10 $      S08AN199900
                       ≠≠STATUS OF THIS ADJUST≠≠                    S08AR200000
    ITEM  ≠AENGIN  S 4 V(MAIN) V(SEC) V(RV1) V(RV2) V(RV3) V(RV4)   S08AO200100
                   V(RV5) V(RV6) V(AMAIN) V(RMAIN) V(RSEC) 0 12 $   S08AO200200
                   ≠≠ENGINE TYPE≠≠                                  S08AO200300
    ITEM  ≠ATIM    F 1 0 $ ≠≠TIME, FL MIN FR OFFSET≠≠               S08AN200400
    ITEM  ≠ADUR    F 2 0 $ ≠≠BURN DURATION, MINUTES≠≠               S08AN200500
    ITEM  ≠AWT     F 3 0 (SLUGS,SNAILS,SLUGS) $  ≠≠TOTAL VEH MASS  ≠≠S08WS200600UP C
                                    ≠≠SNAILS=(LBS.MIN2/E.R.)       ≠≠S08WS200700UP I
    ITEM  ≠AFFR    F 9 0 (SLUG≠MN,SNAIL≠MN,SLUG≠MN) $ ≠≠FUEL FLOW    S08WS200800UP C
                                                  RATE             ≠≠S08WS200900UP I
                                                                                  b
```

Figure 3.—Program interface data (excerpt from COMPOOL document).

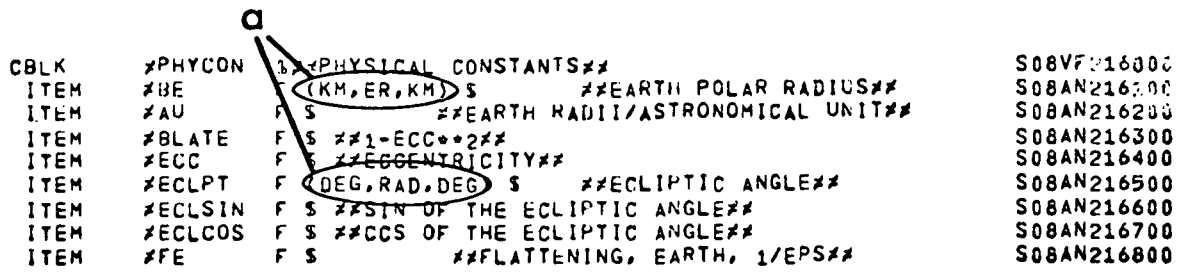

Figure 4.–System constants information (excerpt from COMPOOL document).

Figure 4 shows an example of the system constants information (CBLK), which is similar in form to the VBLK interface data (fig. 3). The C means constant: If a program references an item in this block, it is automatically loaded by the system executive each time the program is operated. The set of three units in the item definition (examples noted by *a*) indicates input, internal, and output units, respectively, for this item. These are nominal and can be overridden on the program request cards.

It should be recognized that COMPOOL does not contain any system data itself. For instance, the PHYCON block in figure 4 does not actually give the values of the constant, but only the description. The constants will ultimately be placed in a block by the input of their values and names (e.g., 10.25 and OMEGA, for Earth rotation rate) into a utility program that uses the assembled COMPOOL to determine the proper block, format, and units for the items. The PHYCON block would then be stored on tape for later use by the operational programs.

The operational programs will not use COMPOOL once they are compiled, however. When a program is first written, COMPOOL items are referenced by name. When the compiler encounters one of these COMPOOL data (or program) references during a compilation, the COMPOOL assembly tables are consulted, and the proper machine code is inserted for manipulation of the data. No computer program in the system, then, contains any interface data definitions, and the program does not directly reference any COMPOOL definitions, it references only the data names.

A number of controls insure the integrity of the COMPOOL document. First, the overall design of the system rules against the inclusion of programs in the system that are not included in COMPOOL. For instance, both the executive and compiler print error messages if a program is used that does not exist in COMPOOL. Second, interface data items cannot be used at all if they are not in COMPOOL before compilation. (However, test and development COMPOOL's may be used before formal submittal of the program for inclusion in the system.) Third, the configuration management system, discussed in the next section, insures that all programs and data in the system are properly entered in COMPOOL. Consequently, the COMPOOL document is a current, thorough, and accurate guide to the descriptions and calling sequences for all system programs and the descriptions and formats of all interface data. COMPOOL and the COMPOOL document simplify program access, use, and maintenance in several ways:

The document is always current because it is produced automatically with each COMPOOL update; it does not depend on the update being done manually. (The actual document production is discussed in the section entitled "Automated Documentation.")

Users can quickly ascertain whether a system program already exists to meet a certain requirement and, if it does and can be called by another program, what calling parameters are necessary. The document excerpts discussed in this section indicate the various COMPOOL entries and the amount of information that is available to the users.

Availability of the COMPOOL document helps maintenance and development personnel decipher portions of programs that reference COMPOOL data. Also, well-documented and accessible data definitions aid considerably in the production of new and modified programs.

The ability to reference data by name instead of having to include actual values in a program eliminates many mistakes and insures consistency in program results. This, in turn, aids in keeping the program and documentation in close harmony.

Constant data can be changed in one place, and all program references are automatically made to the new value. This is because the data can only be called by name in the program. The compiler converts this name to a location reference, and all references are made to this single location. The actual value of the constant need never be stated in the program, it need only be given in the library tape document (described in the section entitled "Automated Documentation"). The automated documentation of constants in a single location, then, is made possible by COMPOOL.

COMPOOL table or format changes require the recompiling of the programs that reference the altered items. However, programmers are not burdened with manual modification of data and definitions in the several programs that may require recompilation. All such modifications are accomplished automatically by the interaction between the COMPOOL and the compiler. Thus programmer errors are eliminated.

The next section will show how the configuration management scheme for the AFSCF software interacts with the software to keep the system user's information sources current and accurate.

CONFIGURATION MANAGEMENT

The COMPOOL document and COMPOOL itself are extremely valuable aids in the standardization and use of the software system. They offer no guarantee, however, that programs will conform to all system standards or that other program documentation will be adequate. To insure that deliverable products are complete, correct, and consistent, checks on product development, adequacy, and compliance with schedules and standards and balances among the skills, methods, and resources available to do the job effectively and efficiently are needed. In AFSCF operations, these checks and balances are provided by a configuration management system.

Configuration management refers to the planning, direction, and control of all factors affecting the state of the system. Some examples of configuration management tasks in the AFSCF system are new program scheduling, review of inputs and determination of COMPOOL content, determination of programs to be included on the master tape, and quality

assurance through the critique and testing of new programs and documents. The balance of this section explains who fulfills the configuration management role in AFSCF, how products flow through the check and balance system, and what some of the specific controls are that provide for quality documentation.

Agencies and Responsibilities

The customer, while having ultimate authority over the system configuration, lacks adequate manpower to directly manage the overall effort. At the same time, the program production, or associate, contractor is too involved with costs and schedules to provide the necessary unbiased support, particularly in the area of quality assurance. Consequently, a third party, known as the integration contractor, is employed to support the customer's configuration management efforts, particularly in the quality assurance and product acceptance areas.

For the past 9 years, SDC has fulfilled this role for the AFSCF software system. The specific tasks performed in this integration role are the detection, definition, and resolution of interface problems (between contractors, programs, and hardware); checking individual programs for conformance to design specifications, standards, and conventions; integration of the programs into a complete computer program subsystem; and validation of the subsystem to insure that all elements of the package are operational and compatible.

Product Development

The checks and balances, then, are between the integration contractor and the associate contractor. Figure 1 delineates the roles of these two contractors as new computer programs pass through the AFSCF product development cycle. The configuration management scheme, based on U.S. Air Force Exhibit 61-47B, was designed by SDC specifically for AFSCF. The following are some aspects of this scheme that aid in securing quality documentation and a quality software system.

The mathematical development for a program is presented in the program design document (step 1, fig. 1). The integration contractor has time to evaluate and digest this information (steps 2 and 3, fig. 1) before the actual program release. The integration contractor also reviews the design document to insure that provision has been made to place new interface data elements in COMPOOL, to properly use the existing COMPOOL information, and, in particular, to insure that no data are imbedded in the program that should be COMPOOL definitions.

When the integration contractor receives the program (step 4, fig. 1), it is manually compared with the programming and design documents. The two products must conform before the program is accepted. This conformance is also required for design logic or any other portion of the design document that could cause problems in later development or analysis work. Programmer analysts rather than computer programs are used to perform these checks. This work is done manually for the simple reason that although the automatic flowchart program can track any arithmetic and logic, no automated method exists that will deduce mathematical derivations or the logical bases of certain techniques from the program (computer code) itself.

The conformance between programs and the COMPOOL items they use is imposed by the system itself. The COMPOOL inputs are prepared from the design document (steps 2 and 5, fig. 1), and the program is then compiled with this COMPOOL. This procedure automatically forces conformance between the program and COMPOOL and also guarantees that the program matches the design document (in the COMPOOL area), the design document being the source of COMPOOL inputs. It follows, then, that COMPOOL, the COMPOOL document, the program, and the design document are all consistent.

Quality Control Measures

The system standards and conventions are enforced throughout the review of the documentation and the final program acceptance. There is a single standards and conventions document for the AFSCF orbital prediction and command system. Briefly, some of the areas covered by this document are–

(1) COMPOOL inputs–conventions, format standards, necessary description information, block sizes, etc.
(2) Data cards–format standards, use of special columns, error processing conventions, etc.
(3) Documentation–format and content standards, program calling sequence, illustration conventions, etc.
(4) Executive interface–illegal instruction standards, input/output (I/O) usage, program size requirements, successor call and nesting standards, compiler usage, etc.
(5) I/O usage–choice of units, access methods, record sizes, internal tape label information, lines per page, and heading information required by system
(6) Messages/error detection–requirements for error messages, error message output devices, system error messages, etc.

This list is only a sample of the areas covered; however, it indicates the extent to which the system is governed by standards and conventions. How these restrictions and guidelines aid the documentation task can be illustrated with an example of input parameters for time. Suppose that the time "1330 hours, 59.2 seconds, 3 June 1970" was an input parameter on a data card. The possible variations in input format for these data are almost innumerable. Some of the possibilities are:

3 6 70 13 30 59.2,

3 6 1970 1330 59 2,

3 JUNE 70 13 30 59.2,

JUNE 3 1970 1330 59 2

Definition of a standard input format for this example accomplishes several things. The program documentation is easy to read, the interpretation of each of the parameters comprising the time is unnecessary because there is only one format, and typographical errors are of little concern because a single format removes all ambiguity or misunderstanding. Users can quickly format data cards without fear that a particular program does things a little

differently. Finally, a common system program can be used to convert the time input to different reference bases, thus relieving the programmer of a task that would have to be tested, with both the test and program logic requiring documentation. The development and enforcement of system standards, then, can be a great step forward in the solution of documentation problems. The application of standards, incidentally, is generally enforced by the program system as well as by the configuration management scheme. This is because many of the standards have to do with interfaces among existing system programs; violations in these circumstances generally result in error messages and unsuccessful computer runs.

When a new or modified version of a program is delivered to the integration contractor, that program must have a unique identification to differentiate it from all other versions of the same program. That identification is known as the "mod." The mod identifier is compiled with the program and is automatically transmitted whenever the program is loaded onto the master tape. Because the configuration management scheme "freezes" the program upon formal submittal to the system (by directing that the programs be stored on specially controlled tapes), all listings bearing the same mod number are guaranteed to be identical and are an accurate reflection of the program bearing that mod on the master tape. Similarly, the master tape has an identification that is printed out on all computer runs, logs, etc. This identification changes whenever a change occurs in the master tape.

Automated documentation is produced and maintained for each version of the master tape. This documentation shows the exact contents, including program mods, of the tape. Because each configuration of master tape and program is unique and because each is covered by documentation, all guesswork concerning the identification of a configuration undergoing maintenance or troubleshooting is removed. The high degree of interdependence among the system programs in the AFSCF system makes knowledge of the correct configuration particularly essential because different versions of the master tape may be current at the same time in support of different projects.

Many phases of configuration management, such as change control and scheduling, are not discussed here, not because they are unimportant to documentation, in fact, they are quite important, but because their importance is somewhat less tangible and more difficult to explain. The aspects that are presented, however, show how a strong quality assurance endeavor, backed by software expertise and well-defined standards and procedures, can greatly improve the quality of system documentation.

AUTOMATED DOCUMENTATION

The AFSCF automated documentation touches on all elements of the system: COMPOOL, master tape, library tape, configuration management status information, and all the computer programs. Figure 5 shows these elements and the inputs and outputs that comprise the automated documentation scheme. A major factor in the functioning of the automated documentation is the centralization of data: the master tape contains all the programs and configuration information, the library tape and COMPOOL contain data normally found only in the individual computer programs, and the status information provides complete details for the three areas of status shown in figure 5. Along with the centralization of data, of course, the accuracy of the documentation depends on the use of the strict configuration management methods described in the previous section.

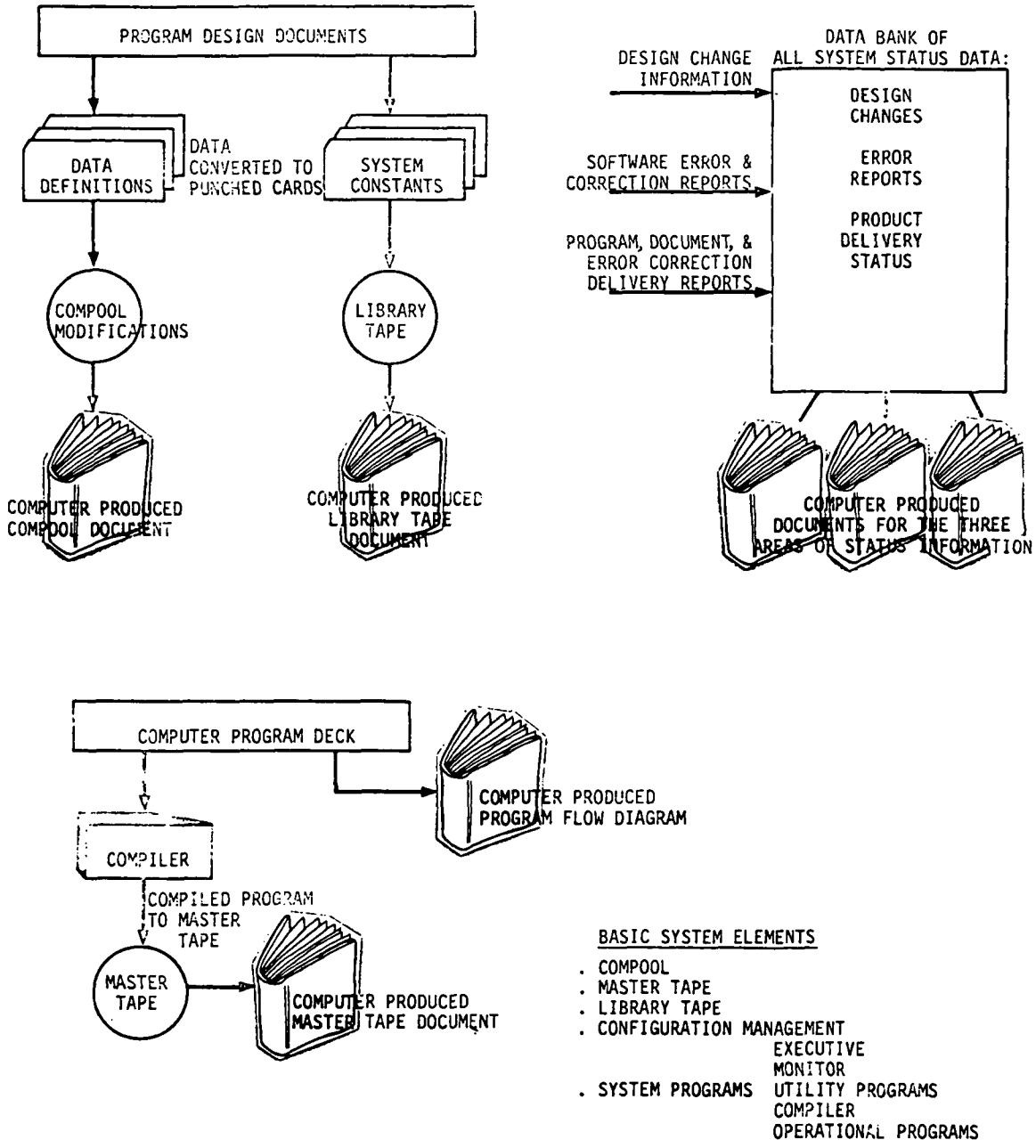

Figure 5.—Automated documentation scheme.

The documents depicted in figure 5 are not necessarily useful as separate entities but are quite useful when taken together as a system document. For instance, the library tape document provides the names and values of system constants, but the COMPOOL document provides the actual description of the data. The master tape documentation is supported by the configuration management data base, which describes significant features of the new program mods and the status of all errors and corrections affecting the tape. COMPOOL and library tape documents and the automatic flowcharts complement the manually produced mathematic and logic documents of individual programs. The balance of this section describes some of these computer-produced documents.

COMPOOL Documentation

The actual production of the COMPOOL document is governed by a program that accepts the list tape (effectively a card image tape), places heading information on the page, and then prints approximately 50 lines of card images. The heading information is requested by a program control card and consists of date, document name and number, starting page number, etc. This program is general purpose; it will accept all AFSCF system list tapes as input and will allow sources of input to be alternated so input cards can be used to annotate information coming from the tape. Figure 6 is an example of the heading output of this program.

Library Tape Documentation

In most systems, the library is the respository for programs, but, in the AFSCF system, the library tape contains the system constant information. The basic form of the input to the library tape program is punched cards, with a list tape being produced along with the library tape itself. Both tapes are processed by the same program that produces the COMPOOL document. Figure 7 is a sample of the library tape document. All the element names are COMPOOL entries, and the sample shown here conforms with the COMPOOL entries in figure 4.

Master Tape

The master tape contains all the computer programs in the system. During the compilation of a program, the compiler sets up tables containing information on all COMPOOL references (both programs and data). These tables are then transferred to the master tape when the program is loaded onto the tape. The program that documents the master tape references these tables, and, along with a log of the programs on the tape, it can produce complete set/use references for all programs and data items. This output can then be placed in the individual program documents (see fig. 1) for subsequent use by the system users and maintenance personnel. Samples of the master tape documentation are shown in figure 8.

System Status Documentation

Figure 5 shows the three areas of status information that are handled by computer documentation. The types of data in the data bank include

2 of 3

Figure 6.–Page heading produced by automated documentation program.

```
ELEMENT  #PHYCON/G                          $                ADQ  062700
         #BE              0.6356775E+4      $                ADQ  062800
         #AU              0.23454710E+5     $                ADQ  062900
         #SLATE           0.9933054E+0      $                ADQ  063000
         #ECC             0.81820E-1        $                ADQ  063100
         #ECLPT RAD       0.409206212       $                ADQ  063200
         #ECLSIN          0.397881208       $                ADQ  063300
         #ECLCOS          0.917436945       $                ADQ  063400
         #FE              0.298250E+3       $                ADQ  063500
```

Figure 7.–Values of system constants (excerpt from library tape document).

(1) Design changes–descriptions of all proposed design changes, identified by control numbers. Status conditions include accepted for future implementation, rejected, and pending action.
(2) Error reports–descriptions of all reported errors in the AFSCF software system, identified by control number and program or by document number, priority, responsible agency, etc.
(3) Product delivery status–description of all program, document, and program corrector deliveries, identified by control number, delivering agency, applicable error report numbers for correctors, etc.

Special report-generating programs employ these data for regular status reports and user information documents when new master tapes are released to the customer. These programs can produce reports with data sorted by control number, status (open or closed problems, scheduled or rejected design changes, etc.), program names, priority of problem, etc. The programs also compile status summaries for the three areas of information. Sample outputs are shown in figure 9.

Flowchart Documentation

The individuality of programmers affects flowcharts more than any other portion of a program document. Although symbols can be standardized, the level of detail is very difficult to regulate, flowchart accuracy is almost as difficult to monitor as program accuracy, and the flow invariably reflects what the programmer wants the program to do and not necessarily what it does. An automatic (computer) flowcharter overcomes all these problems; it is consistent in level of detail and reflects exactly what the program does. Furthermore, it is available as soon as the program is available, which is much more timely than the typical manually produced diagram.

```
                                  93

        P-13.1-D          MASTER TAPE DIRECTORY    PAGE   7

 NAME      TYPE CLASS  LENGTH          ID MOD CONSYS  AJXCOM    DATE

≠DROP      PROG      70000 (732257)  R25SB     WS              18JUN70

≠DROPOUT   PROG      00000 (553551)  R43SG     WS               2JUL70
               ENTRANCES     ≠DROPUT1

≠DRTE      PROG      06060 (050711)  R43SG     WS               2JUL70
```

LOG OF MASTER TAPE CONTENTS

```
≠DAMR           ELEMENT       ENVIRONMENT

        ≠AUDBLK    I*              ≠FIXFLT                        ≠ICP      (≠READ     )
        ≠DAMP                      ≠FLTFIX                        ≠ICP      (≠SKIPF    )
        ≠DATIME    I               ≠GETMDS    I                   ≠ICP
        ≠DATU      I               ≠INTEXP    I                   ≠MVSTST
        ≠DRAWT   (TBLK) *S         ≠IDAA     (IBLK) *U            ≠NUMCON    I
        ≠DRTIME  (TBLK) *S         ≠IOP      (≠CHECK   )          ≠PHYCON  (CBLK)   U
```

ENVIRONMENT LISTING FOR PROGRAMS

```
≠IAN                 IS REFERENCED BY

        ≠DEPLOY                    ≠DOPE                          ≠DRTE
        ≠DETR                      ≠DORSEL                        ≠DRTFG
        ≠DIVERSE                   ≠DOWN                          ≠DRTK

≠IAPEIO              IS REFERENCED BY

        ≠SYSRES

≠IATMOS (TBLK) *  IS REFERENCED BY

        ≠DAD       U               ≠DEKSEL  U                     ≠DIRE     U
```

REFERENCES TO PROGRAMS

Figure 8.—Excerpts from computer-produced master tape documentation.

The AFSCF system has an automatic flowchart program (FLOW) that is currently applied to a majority of the system program documents.

FLOW is designed specifically to work with the system compiler (JOVIAL) and to analyze JOVIAL language statements. It will recognize direct, or machine, code, but it merely sets these off in a box on the diagram.

FLOW accepts prestored tape or card decks (of the object program), and a printer then outputs the flowchart. A more ideal output device for a flowcharter is a plotter, but there

```
                    **** SUMMARY OF DRF#S - CLOSED ****

 DRF    PROG.NAME  MID-MOD  PRI-    STATUS  SUBSYS.  ORIG.  DRF DATE  S/U DATE   MON     DISPO-     PROG.NAME  MID-MOD  MTM   **CHANGED DATA
 NUMB.    DRF       DRF    ORITY                      CO.                        CLOSED   BITION       S/U       S/U    NUMB.  ***NEW ENTRY

 Z0659  #SPRF      P61VC   LOW     CLOSED   E/U      SDC   11/04/68  07/20/70  NEW   P13.1 PEND  #SDARE     P64WC   T00    *
 Z1291  #CARDIO    P74WA   MED     CLOSED   E/U      SDC   08/07/69  07/20/70  NEW   P13.1 PEND  #CARDIO    P74WB   T00    *
 Z1448  #TAPEIO    P73WD   LOW     CLOSED   E/U      SDC   09/22/69  07/16/70  NEW   P13.1 PEND  #TAPEIO    P73WE   T00    *
 Z1582  SYMON              LOW     CLOSED   E/U      SDC   11/13/69  07/20/70  NEW   P13.1 PEND  #SYSRES    Q10WE   T00    *
 Z1593  #DEPLOY    T210C   LOW     CLOSED   OPS      SDC   11/20/69  07/14/70  NEW   P13.1 PEND  #DEPLOY    T21SE   T00
 Z1594  #DEPLOY    T210C   LOW     CLOSED   OPS      SDC   11/20/69  07/14/70  NEW   P13.1 PEND  #DEPLOY    T21SE   T00


                                SUMMARY INFORMATION

      *** DRF#S ON OPERATIONAL SUBSYSTEM CLOSED SINCE 8/18/70 =    27  ***

     HIGH PRIORITY =     1
       Z2280

     MEDIUM PRIORITY =    13
       Z2023  Z2050  Z2118  Z2138  Z2148  Z2147  Z2163  Z2166  Z2172
       Z2182  Z2235  Z2238  Z2291

     LOW PRIORITY =    13
       Z1889  Z2013  Z2022  Z2148  Z2159  Z2171  Z2234  Z2237  Z2253
       Z2267  Z2271  Z2292  Z2312


                                SUMMARY INFORMATION



#DECOR          076  RD  E05                          Z1643          02/04/70
     OCTAL CORRECTORS TO #DECOR TO CORRECT THIS DRF. THESE OCTALS SUBMITTED
     12-10-69 WILL CORRECT THE PROBLEM ONLY AFTER A NEW COMPOOL IS BUILT TO
     INCREASE THE NUMBER OF BITS IN #PASREV. A CCR HAS BEEN SUBMITTED.
     S/U 02/04/70 DRF Z1643 #DECOR   E05 076 RD MTM SUBMITTED
          EXPLANATION ACCEPTED. CLOSURE WILL BE POSTPONED UNTIL INTEGRATION
          OF CCR.

                        STATUS FOR SPECIFIC ERROR REPORT
```

Figure 9.–Excerpts from computer-produced system status reports.

are no plotters available in the AFSCF hardware inventory. Examples of the construction of various flowchart symbols made with printer characters are shown in figure 10.

FLOW can produce flowcharts at several levels of compression. The initial level is approximately one box per input statement. Starting at this level, a set of seven rules is applied to collapse the diagram. The amount it can be collapsed depends ultimately on the logic of the program. In any case, the prime value of this capability is that the amount of collapsing can be controlled at many interim points between the first and ultimate

levels. Thus, flowcharts can be produced that are shorter than the initial level but are still at a level of detail that fully reflects the program logic.

FLOW produces the charts in a form that is suitable for inclusion in a document. The output is small enough that it is suitable for the conventional 8½- by 11-in. page, and all heading information, such as document number, page numbers, and date, are provided.

The program offers options for processing only portions of a program; complete statements in JOVIAL can be output rather than their translations, etc.

The basic advantage gained by the use of FLOW (or any flowcharting system) is that it permits timely and accurate logic flows of the program to be made. In the AFSCF software system there is the additional advantage of being able to cross-reference FLOW output

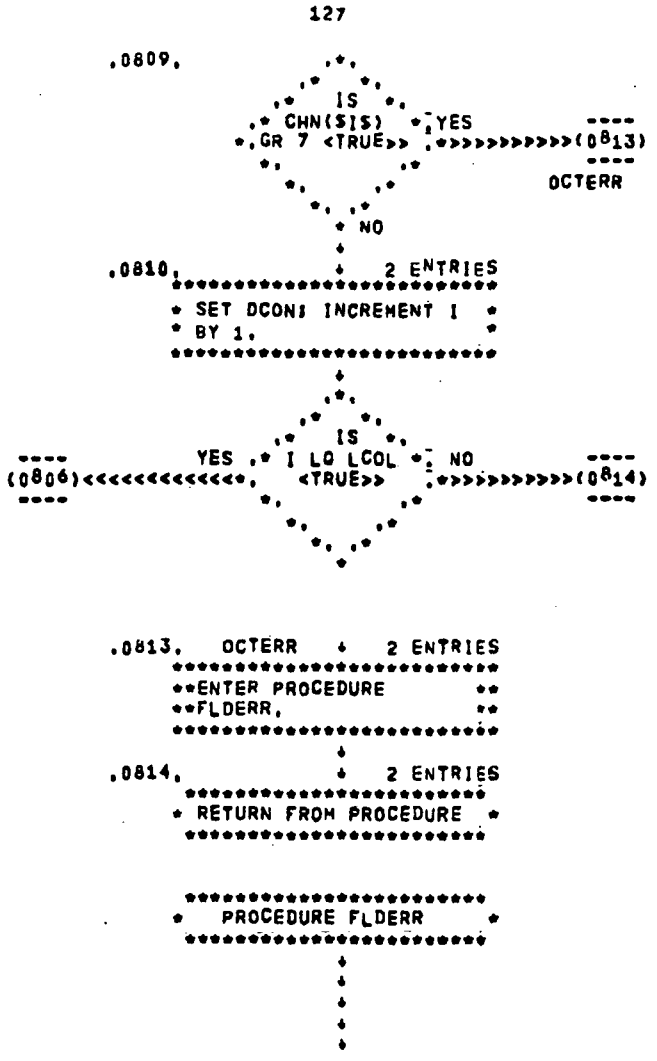

Figure 10.–Automatic flowcharter output.

with the library tape and the COMPOOL documentation. This provides the equivalent of an annotated flowchart with considerably more information content than is available from flowcharts of self-contained programs.

CONCLUSIONS AND RECOMMENDATIONS

The benefits and advantages of the AFSCF system, thorough, accurate, timely, and automated program documentation, are features that would be desired by the users of any system. The question of how to relate the design concepts in this paper to other systems can be considered with the following three facts concerting both this paper and the AFSCF system.

First, the purpose of this paper is not to show specifically how a system should or must be designed but rather to show what can be accomplished by integrating documentation into the basic design. It is unlikely that the same design would be the best approach in any other system, but certainly the principles of design, such as centralization of data, rigid control of the configuration, and program-imposed standards, are valid in other systems.

Second, the AFSCF system was designed under ideal circumstances in that the compiler, executive, monitor, and configuration management techniques were all part of the design effort. This is a tremendous advantage over having to develop a system around already existing compilers and executives, the most common approach to system design.

Finally, the cost of a potential error is so high that AFSCF invests very heavily in "insurance" procedures. The rigid configuration management controls described in this paper are a good example of that. Certainly, it is expensive to critique and review every document and intensively test every program brought into the system. Controlling the master tape, library tape, and COMPOOL is also expensive, but the cost of losing one satellite because of software errors makes the error "insurance" an excellent investment. There are some systems for which errors may be less costly and would not warrant the type of procedures that AFSCF employs.

In spite of the specialized aspects of the AFSCF system, the design concepts are valid for any system. For instance, the centralized data approach is highly desirable, but it is not necessarily practical to implement this approach in an already existing system. However, a data base containing all the desired information for documents can be compiled by requiring that information be incorporated in each program in the form of "pseudodata" (information not required by the compiler, such as comments). These pseudodata must not affect operation of the system compiler. A preprocessing program could then be used to extract this information for documentation. Further, setting standards for the pseudodata and incorporating legality checks on these standards in the preprocessor would help one secure information with a minimum of manual intervention.

Standards and conventions are of value even if they are not rigidly enforced; this is particularly true for documentation content and input formats. The mere fact that programs are produced under standards can provide the user with a good deal of information, even without the use of any specific program documentation. Furthermore, programmers will generally conform to reasonable standards and conventions if they are available. The real interest in automated documentation is not the production of documents; it is making

information available, and standardization can help make information available with fewer documents.

Finally, whether the system is large or small, designed from scratch or only added, the key point is to plan the documentation and not let it just happen. The planned approach, along with some application of the principles presented here, will guarantee better quality documentation for any software system.

DISCUSSION

MEMBER OF THE AUDIENCE: Does the master tape documentation consist of the text or is it a set of module text descriptions?

WOLF: Basically, it is an index, containing program names, dates of loading, number of cells used, etc. One can go from the master tape documentation to the COMPOOL document, which contains the text. The several documents discussed, taken together, constitute the system document.

MEMBER OF THE AUDIENCE: How often do you produce a master tape?

WOLF: At the present time, one is produced every 2 months, although the need for documentation in certain circumstances will occasionally shorten this period to a few days.