An information system for the coordination of program design

by L. BERNSTEIN and F. E. SLOJKOWSKI
Bell Telephone Laboratories, Inc.
Whippany, New Jersey

INTRODUCTION

The best tool for coordinating a large program design project is the computer itself. The programmer is on intimate terms with the computer. He uses it to create his product, to test his product, and to change his product. If he is one of many programmers, all working on a single program system, his activities blend with the activities of others at and within the computer. The very nature of program design is such that an automated management system, built within the object computer[1], can be extremely beneficial in coordinating program design.

The Program Management System (PMS) has been designed with this thought in mind. The PMS is an automated system specifically organized to meet the design, coordination, and management information needs of today's large program production projects. Program designers enter actual design documentation into the PMS as their design progresses and selectively retrieve the documentation of others. The PMS is a tool in-line with the program development process. The use of the PMS today should help establish procedures which will lead to more systematic approaches to program development. This, in turn, will increase the usefulness of the PMS itself.

We particularly have in mind large-scale program development for which "program production" is a better term. Typically, such a project has the following characteristics: the end product is 100,000 machine instructions for a real-time application[2]; 200,000 more instructions are needed for support programs to aid system design, test, and integration; the product is due in three years, when one year of system integration begins; the design organization consists of 100 program designers, 30 engineering analysts, and 30 system engineers; only general system requirements are known at the beginning, and these have been prepared with little thought of implementation.

The program production process

The literature suggests that the approach to such a project is reasonably standard.[3,4,5,6] The procedure is shown in Figure 1. System engineers prepare performance requirements which describe the operations of the software subsystem in light of overall system requirements. Engineering analysts, working with lead programmers, translate these requirements into program functional specifications. In this activity, they identify major program blocks, describe their functions and interrelationships, and impose the first structure on the software subsystem. These same people, with the help of more programmers, next prepare program design specifications. Here, they structure the software subsystem in much greater detail: they define components of the major program functions, and specify equations, algorithms, and data handling techniques.

Next, implementation begins. Programmers translate the equations, algorithms, and data handling techniques into program instructions and describe the impelmentation in a program design description. The first level of testing then takes place as code is debugged and completed blocks of code are tested to prove that the blocks operate as described in their design specifications. As the several blocks which comprise a major function are verified, they are integrated and tested as a function. The success of this test is measured by whether or not the blocks, taken together, properly perform the operations described in the functional specification.

Finally, functional integration takes place. The performance requirements provide the criteria by which proper operation of the integrated functions is judged. Most likely, the major part of this level of integration takes place at the site of the prototype system installation, where all of the major system components, hardware and software alike, are brought together for the first time.

Throughout this overall development process, it is essential that each step produce thorough documentation.[3] Basic system requirements can be constantly changing during program development and such changes can affect the smallest detail of code. Ob-

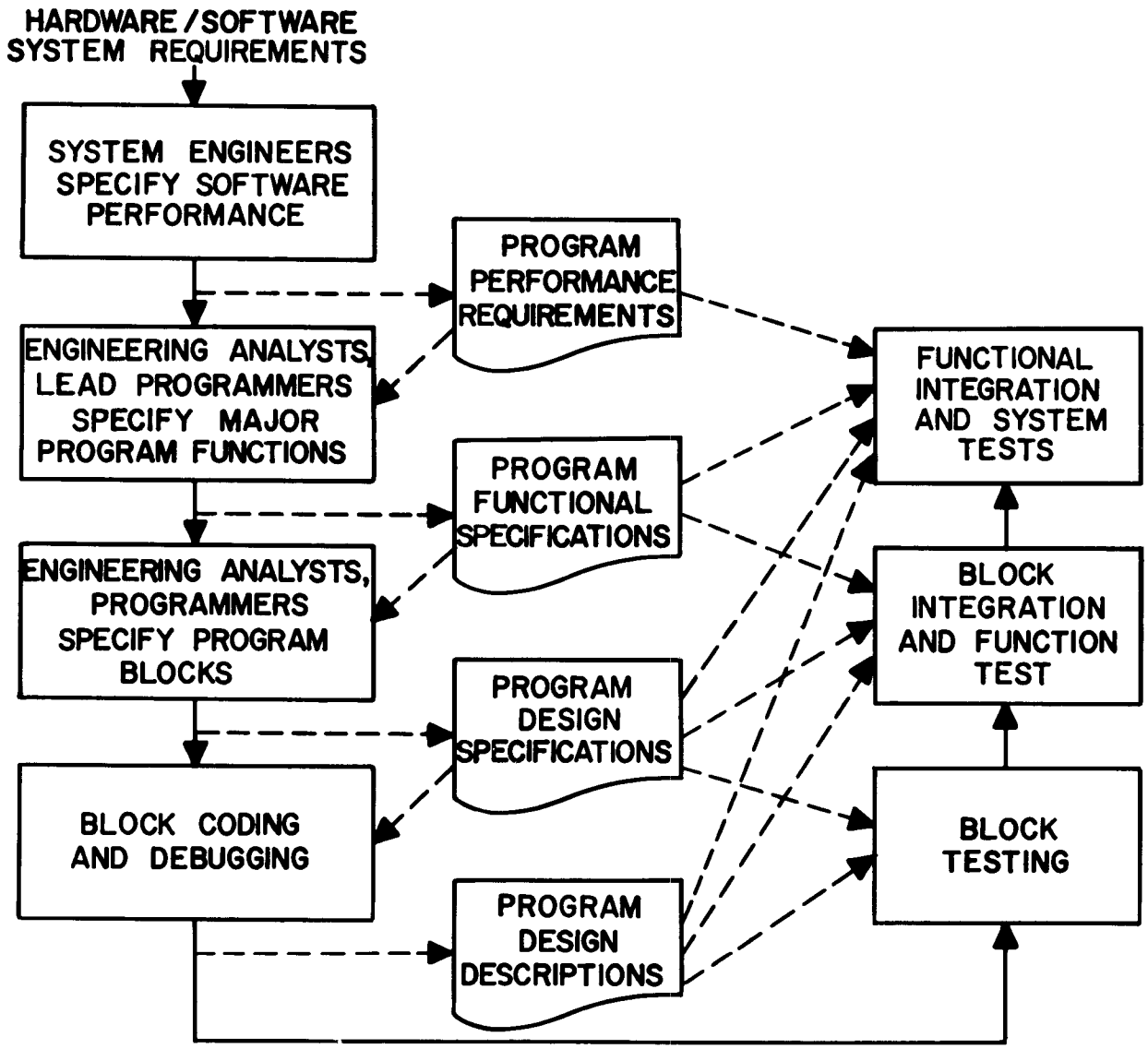

Figure 1 — The program development process

viously, when program changes are required, they will be accomplished much more quickly if complete and up-to-date documentation is available.[6] More importantly, however, fewer changes will be required; for, with adequate documentation available during specification and implementation, better communication will take place between designers. Potential problems will be corrected before they can propagate.

The program development process can be plagued with problems arising from several sources. One main source is the large number of people involved in program development. At the very least, this introduces the problem of discovering and disseminating information about what everyone is doing. At the worst, it creates the problem of assuring that, when a new system requirement is imposed or an old requirement changed, all those whose design is affected are made aware of the change and thoroughly understand it. This is particularly critical in the design of today's real-time systems in which the smallest blocks of code are highly special purpose: a change in system requirements often requires a change in each of five blocks rather than a change in the way a single block uses five library subroutines.

The number of different program development activities taking place at one time introduces further problems. Specification, testing, and integration of various blocks of code often occur simultaneously. In one sense, these activities are separate; if possible, the limited state of development of one block must not prevent progress on other blocks. In another sense, they are highly interdependent; all blocks must eventually be complete and operating together by some fixed date. Somehow, information about these activities must be derived from

documentation and studied from both viewpoints.

Further problems occur when, fostered by the complexity of the program production process, misunderstandings arise between designers. The misunderstanding may concern the smallest detail — the units, scaling, or name of a variable, for example — but the repercussions may be far-reaching.[6] Such misunderstandings, however, are often symptoms of weaknesses in man-to-man communication which must be fostered among designers throughout the project.

Objectives of the program management system

The PMS is intended to provide solutions to these problems in several ways. First of all, it allows the major part of design coordination to be carried out by the designers themselves. In the PMS, designers store documentation on each program block they are concerned with. In their documentation, they identify the type of design activity they are currently involved in — specification, implementation, or testing. They also identify, by name, every other block which, a) "uses" their block as a sub-function, b) is "used by" their block, c) "sends outputs to" their block, and d) "receives inputs from" their block. The PMS automatically creates internal references* between their documents and the documents describing any blocks in these four categories.

In this way, designers impose a structure on the information in the PMS. This structure reflects the structure of the program production process actually taking place. This structure evolves with time as new requirements are imposed on the program system, as new personnel are added to the job, and as design progresses. As a result, the PMS is more like a generalized file system for related data than like an automated library.

Once design documentation is stored in the PMS, it can be relatively easily maintained using automated editing. Moreover, each time a programmer approaches the computer in his day-to-day work, he has at his finger tips all the documents of other designers. Using indices prepared by PMS, he may selectively retrieve any information of interest to him.

But the PMS is to be an active contributor to the coordination and management of program production projects, not merely a passive storage and retrieval device.

Since the information within the PMS reflects the program production structure, the PMS can be an especially powerful tool for aiding management and coordination. Any automated operations performed in the information system are simultaneously carried out across the product space, the functions and programs being designed, and across the production activity space, the designers and their various tasks. In other words, "what is being done" and "how it is being done" are intimately bound together in the information. The PMS makes use of this fact in a variety of ways.

The PMS traces the references between documents to find meaningful subsets of documentation. Traces along the "uses" references locate documents within a nested hierarchy of program blocks. Once these documents have been found, they may be analyzed to obtain answers to particular instances of the general question, "Is the whole equal to the sum of the parts?" Such questions might include, "Are the schedules of the blocks compatible with the schedule of the block they belong to?"; "Is the intended function of the major block reflected in the combination of its component blocks?"; "If I must change the major block, what component blocks may require changes?"; etc.

Traces along the "inputs" and "outputs" references locate documents along an operational path — a chain of blocks which transforms a given set of inputs into a given set of outputs, thereby accomplishing one of many operations which the overall system performs. These documents may be analyzed to obtain answers to specific instances of the general question, "Are there any weak links in any operational chain?" Here, typical questions might be, "If Block B receives inputs from Block A, does the documentation on these two blocks describe the variables identically?"; "If the outputs from a block must be changed, who are the designers of all the blocks which use that output and must therefore be consulted on the change?"; "What level of development has each block in a given operational chain achieved, so I may know when the complete operation may be tested?"; etc.

In addition to these features, the PMS also diagnoses the information it contains. For example, it will compare the description of a common interface as contained in two or more documents. Any inconsistencies found will be reported to the designers involved.

These reference-tracing and consistency-checking capabilities will also be extremely useful to managers of program production. Even though the designers themselves impose the structure on the information, it is management's job to assure that this structure matches some desired objective. The PMS will, in some cases, automatically aid in determining this and, in other cases, will provide access to all the information needed to discover problem areas.

Design requirements on the program management system

The objectives of the PMS, therefore, impose three

*The term "internal references" is used to describe the connections between files containing documents within PMS. These connections are implemented using list processing techniques. The term "references" is used to indicate the relations between the blocks being documented.

basic requirements on its design: 1) an information structure imposed by the users; 2) the binding together, in the information structure, of information describing the product and information describing the activities producing the product; 3) the means for analyzing the information to derive insight into the development activity. Add to these the further requirement that the system be physically configured so that the users may make it a part of their daily routine. Together, these form the basic design requirements on the PMS.

A big step toward the requirement of convenience lies simply in the fact that the PMS is a computerized system: program designers will find it easy to use and easy to find, for it will be in the same computer for which they are designing programs. It will be imbedded in a time-sharing system which itself is being designed especially for use in the program design, test, and integration activities of large program production projects.

This time-sharing system, however, is not yet operational. Therefore, we have built a "simulated" PMS operating in a batch mode on the IBM 7094. This system is providing much-needed insight into both the further design of the PMS and its relation to the user. This "simulated" PMS, shown in Figure 2, is the system which will be described in the remainder of this paper. However, we ask the reader to adopt the same attitude we had when we designed the system — always keep in mind the ultimate time-sharing environment in which the PMS will operate.

Several criteria were important in the design of the batch system. First of all, it was important that the system be operational as quickly as possible. For this reason, we made extensive use of various software

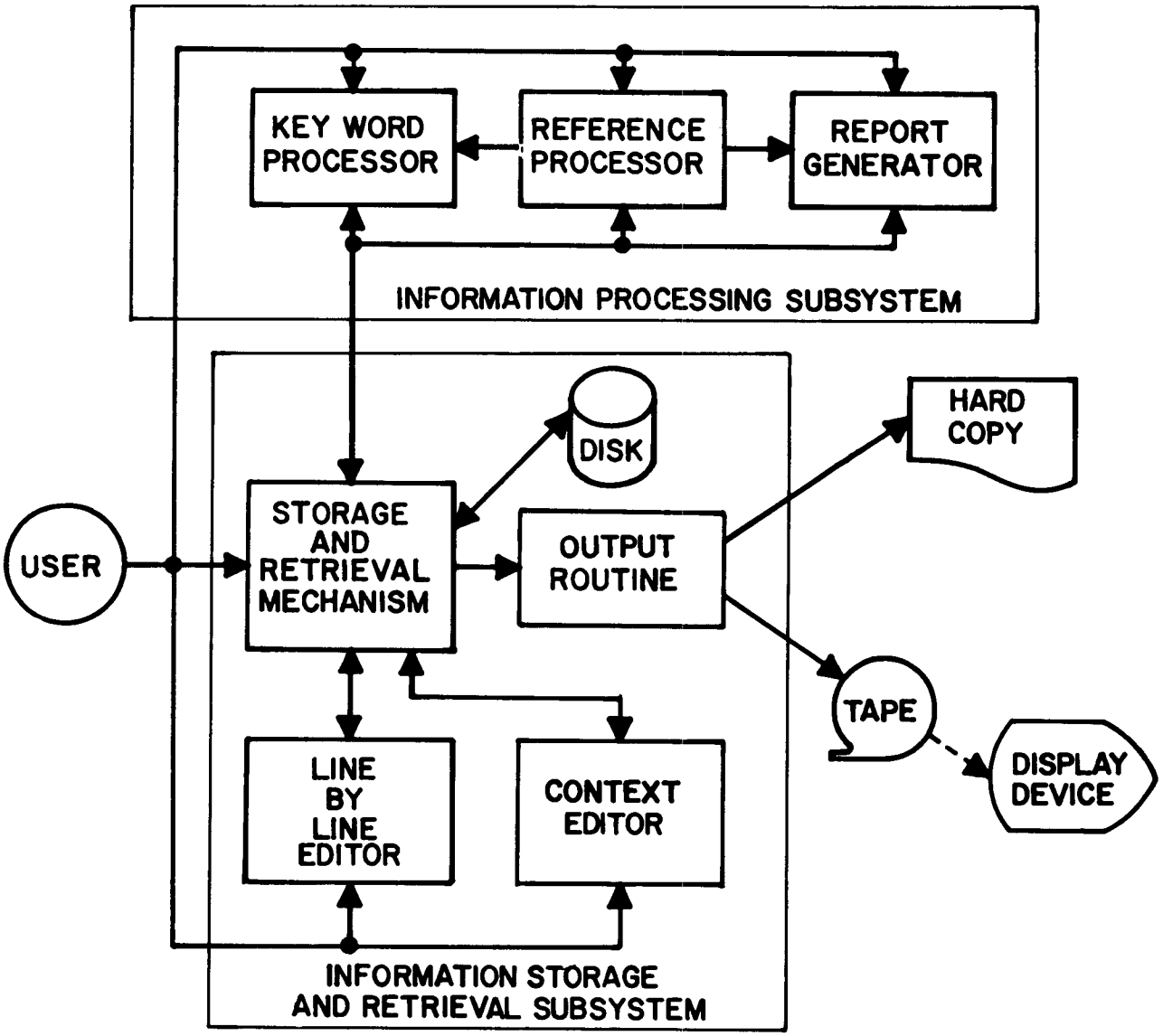

Figure 2 — PMS block diagram

facilities developed elsewhere in the Bell Laboratories. Expediency took precedence over final operational efficiency.

Secondly, we wished to use this "simulated" PMS to acclimate the users to the idea of using automated methods of documentation maintenance and analysis and to test user reaction. Therefore, the use and operation of the batch system is supported by a rather extensive staff operation. The staff personnel shield the users from the sometimes tedious operations necessary in translating information processing requests into actual batch runs. These tedious aspects will disappear under time-sharing and we do not wish to foster antagonism before that time.

Documentation plan

The design of PMS included the specification of a documentation plan which simplifies the process of recording design information and storing it in PMS. This plan identifies the minimum set of documents needed for adequate documentation of a program design effort as:

1. Functional Specifications
2. Design Specifications
3. Design Descriptions
4. Test and Evaluation Documents
5. Work Control Documents

Studies of the information that should be included in each of these led to the realization that a single format would accommodate most of the information needs of the first four. Naturally, this format can be modified where necessary; in fact, the plan is designed so that it can evolve to satisfy new documentation requirements.

Before describing this single format we shall briefly examine the requirements for each document, based on the program production process described earlier in the paper. More detailed descriptions can be found in the literature.[5,6]

Functional Specifications are prepared by systems engineers. They contain a "black box" description of *what* each program block will do. As the name denotes, these specifications translate performance requirements into the functions each program block will perform, and contain a description of its interfaces.

The Design Specifications are prepared by system program designers; they contain descriptions of *how* each block is to perform its assigned function and contain the equations or algorithms the block is to execute.

The Design Descriptions are prepared by the programmers while they are writing their code. They contain the detailed descriptions of calling sequences, data scaling, error recovery provisions, etc., and a well-commented program listing.

The Test and Evaluation documents are prepared by system engineers responsible for test planning. They contain descriptions of subsystem tests and acceptance tests needed to verify and evaluate system performance. These tests involve test programs and procedures which themselves require essentially the same documentation as program blocks.

The Work Control Documents needed for project coordination are prepared from information extracted from all of the above. They contain summary information indicating the status of the overall design effort, as well as a breakdown of the status of the individual program blocks.

The single format document which can be used to satisfy the needs of each of the first four documents contains the following general items:

1. Title — This is the name of the system block being specified or described. It must be unique among all system blocks. This requirement is imposed both by system coordination and PMS design considerations.
2. Heading — This item contains a variety of entries identifying the person responsible for the design and key descriptors of the block or document. One such descriptor is the type of document which would be either Functional Specification, Design Specification or Design Description.
3. Purpose — A one or two sentence narrative stating the purpose of the block within the system.
4. Description — Approximately a 500 word discourse on the functions the block performs.
5. Inputs — The data and/or control information the block receives and the name of the block where it originates.
6. Outputs — The data and/or control information a block provides and the name of the block where it goes.
7. References — The connectivity information identifying the relation of this block to other blocks in the system as well as references in the literature. As indicated earlier, one set of references reflecting the program production structure consists of "uses" and "sends outputs to" and their inverses "used by" and "receives inputs from."
8. Diagrams — Block and/or flow diagrams describing the block operation.
9. Block Test Plan — An account of how this block will be or was tested as a separate unit. It would include the test procedures and test results necessary to verify the operation of the block.

To see how this single format can be applied to these varied documents let us consider the Inputs item. In a Functional Specification, this item would contain the

names of the other blocks sending data to the one being documented. If the inputs are known explicitly to the system engineer, he would list them along with the accuracy, rate of data availability, etc. However, the system engineer usually only identifies the class of data being transferred between blocks, so this is what he would list. The Design Specification would explicitly identify each data and control variable along with its source, precision, rate of availability, etc.

Then, the Design Description would contain the detailed description of the variables including their format, scaling, operand memory location and packing. If this information is described in a data set description, the Design Description for the program would identify the data set and the variable name. Note that in this case a Design Description would be written for the data set as well as the program.

The total documentation of a program block can be viewed as the combination of these three documents. While they are usually prepared by three different persons, it is useful to produce one single document containing the information written by all three. As shown in Figure 3, PMS provides the capability to produce this single document.

To simplify the designer's task in originating information, he is provided with an outline format, more detailed than the one above, so that he can "fill in" the appropriate items.

At a later date the information user can request PMS to automatically extract specified items from all or some subset of the documents. For example, the system coordinator could request PMS to compile the heading items from all documents to produce a single coordination document.

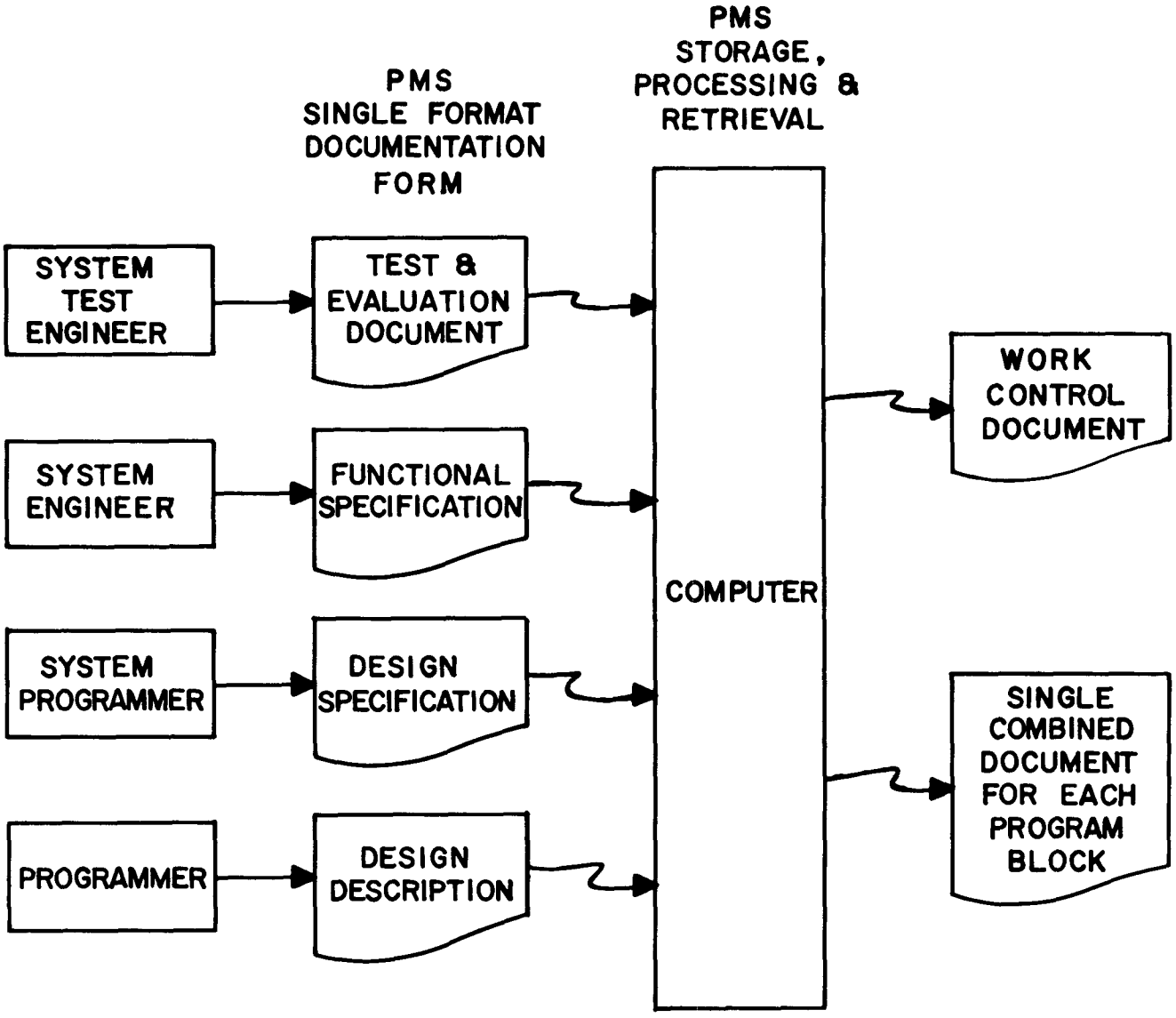

Figure 3 — PMS user interfaces

Functions of PMS

Once documents are prepared using the outline format they can be processed by PMS. The functions PMS will perform are:

1. To store and retrieve these documents.
2. To provide internal references between documents.
3. To process internal references and information to answer questions about the design and implementation of the system as documented in PMS.
4. To format the answer to questions.

Since PMS is a user oriented system it is designed according to the following criteria:

1. PMS must be devoted fully to needs of a user. He should be able to modify and augment the system as his needs dictate and have access to information in a variety of ways.
2. PMS must be simple to use. There should be some meaningful subset of the system which a user can teach himself by reading a manual or operating a user's console in an "instruction mode".** Naturally, in order to use PMS in a more sophisticated manner the user must have more sophisticated knowledge of the system. Nevertheless, there must be a simple method of using the system.
3. PMS must be evolutionary. When new features are added to PMS, service should continue during the installation phase. This implies that new features should be designed within the framework of PMS and in turn that this framework provide for an open ended growth.
4. PMS must provide extensive default mechanisms for all command options and make reasonable decisions wherever possible.

Data organization

The documents are stored in the PMS information pool.[3,7,8] This pool consists of files each containing a document describing a program block. The name of the file corresponds to the name of the block.

The file in turn is made up of subfiles and a set of internal references to other files. Each subfile contains an item of the document as described earlier.

A subfile is an information set which can contain text descriptions, equations, graphical data (diagrams) or tabular information. In general, a subfile will consist of one of these types of information with the exception that equations can be interleaved with text or graphical data. Thus, a subfile can contain any kind of information in a specified format, and it can be as long as desired, within the physical limitations of the storage mechanism.

The internal references are stored as reference sets for each file. They differ from the subfiles in that they express the links between program blocks or files. A reference consists of its type, the two file names involved in the connection and some baggage carried along as a modifier. The baggage facility permits the user to define in more detail the relation between files. For example, if the user considers such characteristics as size of block or operating environment to be important, he may specify these in the baggage and may later modify the results of reference processing based on these descriptors. The user designates references for his block; PMS then automatically creates the inverse internal references and checks to insure that the designated references are consistent with the internally stored references.

File structure

Data is stored in PMS in a two-level file structure.[9] As we have seen, it is composed of a file with its associated subfiles. A file only contains information in the sense that its subfiles contain information. As suggested by Nelson[10], this file structure can be shaped into various forms, changed from one arrangement to another in accordance with the evolving program production activities. This is accomplished by having the subfiles and references be lists associated with the file name. Thus, associated with the file is the file name, a list of pointers to its subfiles and a list of pointers to other files.[11, 12] Internal references between subfiles are not permitted; however, there are as many two-way internal references between files as reference types. The user can augment the structure of the information pool for his own need by defining a new internal reference and applying it to existing files and newly created ones.

The references which reflect the program production structure are of interest to the project as a whole. These reference types appear on the documentation form and pointers are provided within each file for them.

A subfile can contain as much information as desired and any number of subfiles can be defined for a file. The information set can be stored in a subfile in either a free format or a defined format mode.

The free format mode permits the storage of an unlimited amount of text. The defined format mode can be used to define a structure to the extent that the subfile consists of several subitems, each with relatively free format, or to the extent that each field in the subfile is fixed. The more explicit the format definition the more efficient the processing of the subfile will be. In this sense the defined format mode provides a facility for multi-level definition of the structure of a subfile.

**Once the system enters its time-sharing mode of operation, the use of a user's console in an instruction mode would permit the computer to explain to the user via CRT or a typewriter, how he can use the console and give him examples so that he can "learn by doing."

An example of PMS file organization

Figure 4 illustrates this data organization. Program T consists of Routine 100 which in turn consists of the subroutines A, B, C. A data flow diagram for Routine 100 is shown. This diagram would be encoded and stored in the diagram subfile of the file named Routine 100.

The information pool would be organized as shown on the upper right side of Figure 4. Each block represents a PMS file. The references are shown explicitly as arrows and several subfiles for subroutine A are shown.

Organization of simulated PMS

PMS is composed of two subsystems, the Information Storage and Retrieval Subsystem and the Information Processing Subsystem, as shown in Figure 2. These subsystems perform the document handling and processing functions listed earlier. A description of each subsystem follows.

The maintenance, modification, storage and retrieval of information in PMS is the concern of the Information Storage and Retrieval Subsystem. This subsystem consists of the directories used for disk storage, editing program, and a documentation form input and output routine.

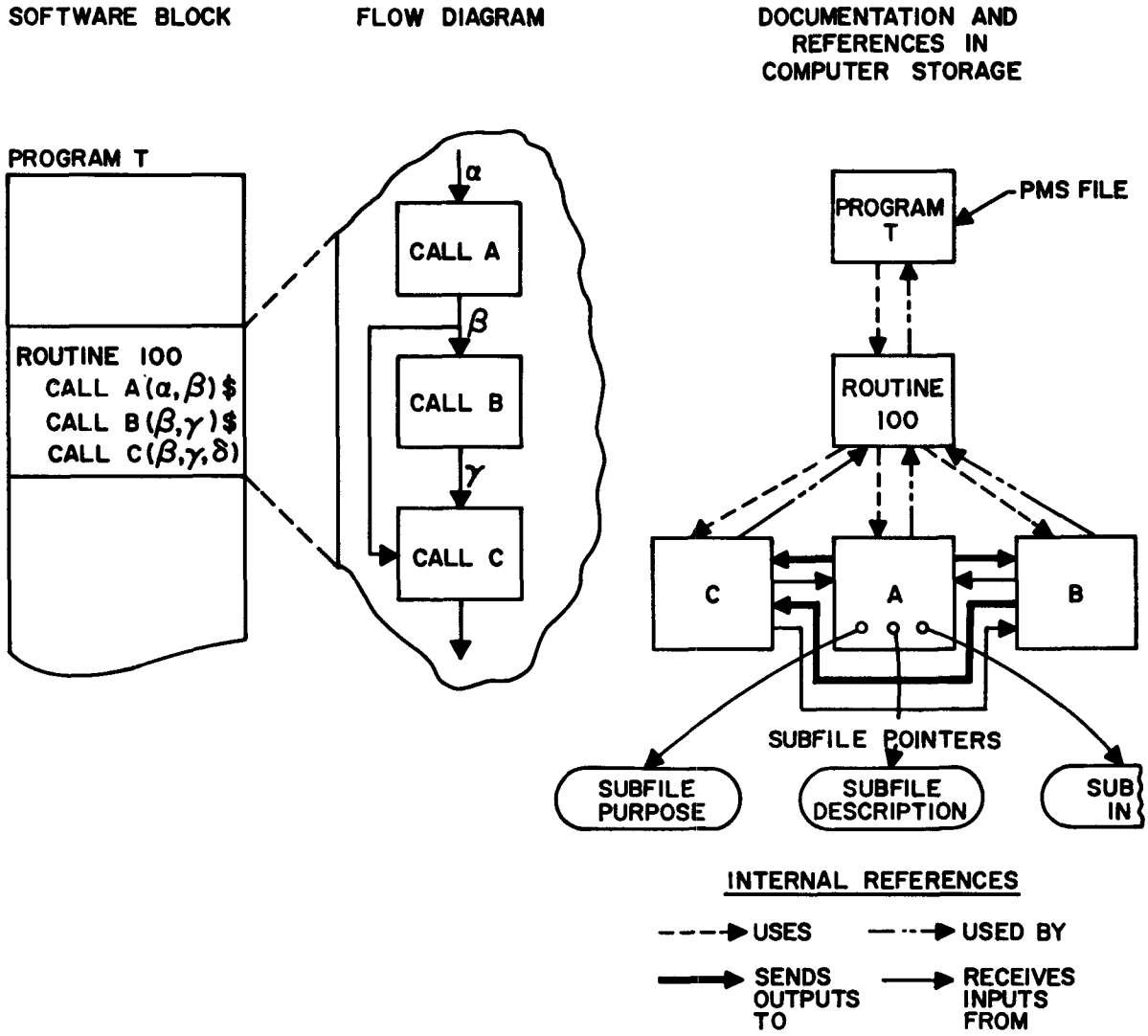

Figure 4 — PMS data organization

A line-by-line editing facility is provided so that the user can change any line containing any type of information or insert lines in his subfile by specifying the line number automatically provided by the system.

A context editing facility is also provided which simplifies the process of modifying text descriptions, since subfiles are primarily used for this type of information. This program finds an arbitrary length character string in the subfile and changes it, deletes it or adds another string after it. This context editor also paragraphs the information and permits editing of equations imbedded in the text. Provisions are made for special layouts; however, every option has a default mechanism designed to follow "natural" conventions so that the user need not be conscious of special conventions when he is creating his information.

Currently, a general purpose output routine will fomat data for output on the Stromberg Carlson Microfilm Plotter or for a high speed printer. Data can also be written on a special tape and used as input to a digital display device. Since the device is not electrically connected to the IBM 7094 we cannot realize on-line interaction with the information retrieval system. However this feature is essential to maximum effective use of the system, and will be provided in the time-sharing version of PMS.

A group of routines for processing the information stored in PMS comprises the second PMS subsystem, the Information Processing Subsystem. Each routine is independent of the others, and can be used independently or in conjunction with the others to derive a result. The two classes of information processing are subfile processing and reference processing.

The subfile processing routines extract information from subfiles and manipulate it to either generate a report or answer user questions.

These reference processing routines comprise the reference processor which allows the user to operate on the internal references and on sets of file names. The operations permit the tracing of reference trees[9, 11, 13] as well as correlating sets of file names. This processor consists of two sections. First it can find the image or inverse image of a set of file names for a particular relation. For example, it can find the set of all file names Y such that

$$Y = \left\{ y_i; y_i = R(a) \right\}$$

where R is a fixed relation and a is a file name. In the example shown in Figure 4, the set Y which routine 100 "uses" consists of subroutines A, B, and C.

Secondly, a means of correlating two sets of file names to produce a third is provided. The operators available are the usual set operators – union, intersection, difference, and symmetric differences. The ability to compare two sets for equality, and to select individual elements from a set is also provided.

To implement this processor, a dynamic storage allocation facility[14] is required which permits the requesting and releasing of memory blocks as sets expand and contract. This feature will even be more essential in a time-sharing environment where several programs reside in core simultaneously. Some limitations in the current reference processor are the inability to specify universes for set operations — the universe is simplicitly the set of all file names — and the inability to index elements within the set. These limitations will be removed during the next phase of the PMS design.

When PMS was applied to a large software system it was found that the users needed a set of indices — these provide an index by document title (file name), an author index, and a generation breakdown or "where used" index. Therefore, a special purpose index program was designed, the report generator, which would operate on the desired information in the file to form these indices. This program can also extract a common subfile (or subfiles) from each file and list it along with the sorted data. Thus, PMS is able to automatically create a set of indices from a set of source documents without manually extracting information from each file. This index routine can also be applied to any selected set of files resulting from a request made of the reference processing routines.

Typical requests.

This section describes how the PMS responds to some typical retrieval requests.

1. What is the purpose of all the subunits of routine 100 which are involved in storage allocation?
   a. Locate the routine 100 file, using the Information Storage and Retrieval Subsystem.
   b. Find the set of file names which are "used by" routine 100, using the reference processor. Y = USED BY (Routine 100)
   c. Locate the *purpose* subfile for each member of Y, using the retrieval subsystem.
   d. Search each *purpose* subfile for mention of "storage" or "allocation." Separate from Y all those that contain these words.
   e. Output each file name found in d. and its *purpose,* using the output routine.
2. What are the subunits of routine 100 which send inputs to subroutine B? Alphabetize these files by title and output the *input* subfile along with the file name.
   a. Locate the Routine 100 file, using the retrieval subsystem.
   b. Find Y = {USED BY (Routine 100) and SENDING INPUTS TO (Subroutine B)} using the reference processor.

c. Alphabetize the elements of Y, using the Report Generator.

d. Find the *input* subfile for each member of Y, using the retrieval subsystem.

e. Output Y with the *input* subfile, using the output routine.

A compiler will be written to translate these requests directly into PMS commands. For now, a request must be manually translated into detailed PMS commands for a result.

PMS environment

Since the application of PMS deals heavily with text descriptions or equations, a string manipulation language[15] and a set of string macros developed by McIlroy[16] has been used to operate on the information stored in the subfiles. However, this does not prevent the user from writing his own program to operate on numeric information stored in the subfile. In the on-line time-sharing mode of operation a string language, having the capabilities of TRAC[17] will be available to the user.

SUMMARY

As an information system PMS principally provides the information supporting the various activities associated with a complex program design project. The complexities of such a system have been explained and the characteristics of an information system supporting such an activity have been identified.

When a user approaches the PMS system he needs help in finding the information he is interested in — a reference processing facility will assist him there. Once he has found the information he wants, it is processed in the way suited to his needs — the information processing facility will assist him here.

The storage and retrieval subsystem provides a means for storage of descriptive information in subfiles, and a means for linking files together in a variety of ways to create reference trees.

The processing subsystem requires a means for accepting stored information as data and processing it according to user specified procedures written in a high level language. Thus PMS provides statements which will

1. Find the file or files containing information of interest using references and detailed links.
2. Extract a subfile from a file.
3. Perform operations on the data.
4. Output the information after editing, sorting and formatting the information as requested by the user. In the case of editing the stored information will be updated and the edited information will be outputted to provide the user with the results of his editing.

REFERENCES

1 J W PERRY A KENT
Documentation and information retrieval
Western Reserve University Press Cleveland Ohio 1957 chapter 6

2 W KEISTER R W KETCHEDGE H E VAUGHAN
No. 1 ESS system organization and objectives
Bell System Technical Journal XLIII September 1964 pp 1831-1844

3 W H DESMONDE
Real-time data processing system introductory concepts
Prentice-Hall Inc. Englewood Cliffs N J 1964 pp. 137-147

4 R J COYLE J K STEWART
Design of a 'real-time programming system'
Computers and Automation vol 12 no 9 September 1963 p. 31

5 R V HEAD
Real-time program specifications
Comm. ACM 6 July 1963 pp. 376-383

6 T A HOLDIMAN
Management techniques for real-time Computer programming
J ACM July 1962 pp. 387-404

7 T HARLOW
Research in information retrieval — An investigation of the techniques and concepts of information retrieval
Technical Report 5400-TR-0096 AD 461099 July 1964

8 R P BARRETT
CIRC — Centralized information retrieval and control
Proceedings of Workshop on Working with Semi-Automatic Documentation Systems AD 620360 AFOSR 65-1699 May 1965

9 E H SUSSENGURTH
Use of tree structures for processing files
Comm of ACM vol 6 no 5 May 1963 pp 272-279

10 T H NELSON
A file structure for the complex, the changing and the indeterminate
Proc ACM August 1965 pp. 84-100

11. D T ROSS J E RODRIGUEZ
Theoretical foundations for the computer-aided design system
Proc SJCC 1963 pp. 305-322

12 C W BACHMAN S B WILLIAM
The integrated data store — A general purpose programming system for random access memories
AFIPS Conference Proceedings 1964

13 J BECKON R M HAYES
Information storage and retrieval: Tools, elements, theories
John Wiley & Sons Inc New York 1963 Chapters 13-14

14 W S BROWN
An operating environment for dynamic-recursive computer programming systems
Comm ACM 8 June 1965 pp. 371-377

15 D J FARBER R E GRISWOLD I P POLONSKY
SNOBOL, A string manipulation language
J ACM 11 January 1964 pp. 21-30

16 J D MC ILROY
A string manipulation for FAP programs
Unpublished

17 C N MOOERS L P DEUTSCH
TRAC, A text handline language
Proc ACM August 1965 pp. 229-246