1-1-1984

Productivity aids for software development.

Lester L. Zern

PRODUCTIVITY AIDS FOR SOFTWARE DEVELOPMENT

by
LESTER L. ZERN

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Computing Science

Lehigh University
1984

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

July 30 1984
(date)

Professor in Charge

Chairman of Department

TABLE OF CONTENTS

Abstract 1

Introduction 2

Productivity Aids 5

Office Automation 5

Design and Coding 8

Testing 11

Productivity Aids for Screen Design 13

Display Terminal Interface 13

Terminal Dialog 17

Elements of a Dialog 20

Formatting of Displays 21

DEC/Forms Management System 23

DEC/FMS Dialog Development 25

DEC/FMS Dialog Management 28

IBM/Interactive System Productivity Facility 30

ISPF Dialog Development 32

ISPF Dialog Management 35

Statistical Analysis System/Full Screen Product 39

Conclusion 45

Bibliography 47

Candidate Biography 49

ABSTRACT

Productivity is one of the most talked about subjects in data processing today. It seemingly is the concern of everyone from the board of directors, to the manager of data processing, to the software designers and programmers. Likewise, there is a new awareness that productivity is more than just volume. It includes the quality and reliability of the software product and the efficiency with which the product is produced.

This paper reviews a class of tools that affect the whole environment of software development: productivity aids. Since software development is becoming more like a software factory, productivity aids play a vital part in increased productivity. These tools are what may be called "captured intelligence". They represent the cumulative knowledge and effort of many people. Various tools have evolved out of a need to improve the quality and consistency of the software system being developed. Software engineering has improved our software development techniques. Productivity aids are one way in which one can implement many of these techniques.

Productivity aids and their application to specific areas of software development are reviewed. After this, a detailed review of productivity aids for display terminal screen design is presented.

PRODUCTIVITY AIDS FOR SOFTWARE DEVELOPMENT

Introduction

In the early 1960s computerization became a growing factor in business. In the same time span the national productivity level, which had been growing at about 3.2 percent yearly since World War II, suddenly started to drop. It bottomed out at less than 1 percent in the late 1960s and has not regained its growth rate since that time. Many factors contributed to the decline in productivity. The computer, as the perfect replacement for many routine jobs, may have accelerated the decline by alienating many workers. However, without the advent of computers, national productivity would be far lower.

With the arrival of computers an extensive set of new careers was established. Thousands of jobs in the manufacture, maintainance, and programming of computers were created. One area of computing that is currently receiving much attention is the productivity of the software development effort.

Productivity is the ability to create a quality software product in a specified period of time with finite resources. The way in which the different phases of software development are completed affects the overall productivity of the software development effort. Typically, this development effort includes a proposal, feasibility studies,

system definition and design, programming, implementation, and maintenance. How we interface with the computer system on which a software system is being developed and what tools are available for our use can affect the ultimate product and the productivity of the system development effort.

The advances in electronic technology have improved the cost performance of hardware well beyond the improved cost performance of software engineering. Today we see that machine cost is the major cost for less than 5 percent of the man-machine interactions. On the other hand, human time is twenty times more expensive than machine time for over 95 percent of the man machine interactions. Current projections indicate that this trend will continue with higher human costs in the future. Therefore, improving these man-machine interactions will improve the overall productivity of the software development effort.

Another need arises out of the problems observed in the software systems developed. Many of them are not responsive to the end user s needs; they do not meet specifications. Their reliability is uncertain and their costs are excessive and unpredictable. Many delivered systems are not easily modified; software maintenance is complex, costly, and error prone. The system is often delivered late and is less than promised. In addition, the process of system developement is too often inefficient and does not make optimal use of the available computer resources.

Software development is becoming like a software factory. Analysts design software systems and programmers manufacture and repair them. Modules are put together like parts of a car. With maintenance costs consuming 50-80 percent of data processing budgets, one can see that the way in which these modules are designed and programmed is very important.

Software engineering has developed new design and programming techniques for software development. High level languages such as PASCAL and Ada implement many of these techniques. There are new ways to interface with the computer, such as light pens, digitizing tablets, touch sensitive screens, joy sticks, optic scanners, and voice. All of these improve our ability to deliver a quality product and improve our productivity.

The software development environment needs to have a coordinated set of tools for software development. These tools should support the entire software life cycle. In such an environment the designers and programmers can effectively communicate thoughts and ideas between themselves and the end user. They will be able to write system specificatons, dorument the software system, develop software models, write the program code, and test the end product. Unfortunately, most environments do not have such a set of productivity aids. Using the computer as a life cycle tool for software development will then have limited surcess.

Productivity Aids

Hardware interfaces to the computer have improved over the years. The standard interface for input and output has changed from cards and printers to display terminals. Some terminals are equipped with touch sensitive screens, optic scanners, and digitizing tablets. Software development has evolved from wired boards to high-level languages. Productivity aids have been developed to take advantage of these changes. They can be applied to every area of software development. The judicious use of these aids can improve the software development process by improving the productivity of software designers and programmers.

These software tools, or productivity aids, represent what might be called captured intelligence. They represent the combined knowledge of many software developers in trying to improve the productivity of software development. Productivity aids include within them techniques and procedures that address specific problems in software development. Listed below are a minimum set of tools that will improve the usefulness of the computer as a tool in the life cycle of software development.

Office Automation

"Word processors" automate almost every kind of typed document. Since software designers and programmers are already familiar with computer terminal keyboards, word

processing can be a useful extension of their keying skills. Instead of writing, they can type their thoughts directly into machine readable form. With the burden of writing simplified the likelihood of better documentation of specifications, system changes, and user procedures will improve. Document preparation can be further simplified by providing skeletons of the documents required. This improves document readability, completeness, and accuracy.

"Electronic mail" improves the communication of thoughts and ideas between designers and programmers. A Boose and Allen study found that telephone calls reach the intended person about half the time. Return calls fare no better. Electronic mail can automatically keep track of mail sent and when it was received and read. The receiving parties can review and read their mail when it best fits their schedule. This reduces interruptions and still allows them to respond in a timely fashion to the more important mail. Using electronic mail, specifications, software problems or other relevant information can easily be transferred between designers and programmers. This will improve the end product by the timely and accurate interactions of designers and programmers as the software product is being developed.

"Information retrieval", using database management systems and English like query languages, can provide quick and easy access to information relevant to the software development effort. This can be especially useful where the

system being developed is large and involves many designers and programmers. A database of specifications, current problems, and other relavant information can be used to keep a journal of the software development effort. The designers and programmers can then access this information by using information retrieval systems. It is important that the database management system used be simple and very user friendly. If not, any benefits gained by having a central repository of information will be lost in the complexity of accessing the information.

Another important productivity aid is "personal time management". In any software development effort there will be meetings between designers and programmers. At times, these people may be unavailable for meetings due to vacation, business trips, conferences, or seminars. At the same time, there will also be meetings with the customer/users to review the software developed. Scheduling meetings to accomodate these people can be quite difficult. If a calendar of each person is online, it becomes a much simpler task. Through normal management hierarchies or security systems, upper management could directly update calendars of subordinates without interrupting already scheduled events. Peers could review each other s schedules which will help in planning future meetings. This will improve the interaction of the software development team and improve the overall productivity of the software development

effort.

Obviously the use of these tools requires a new way of doing business. The gains can be significant but not without investment. It will require time to learn the new procedures and hardware to support the needs.

Design and Coding

There are three major parts to any software system: input, output, and the internal logic. The input is the data or requests needed for the software system to operate. The output is the result of running the system. Thirdly, the internal logic of the system is the instructions required to meet the specifications of the system. In most cases, the internal logic of the system is left to the software designers and programmers to establish. The logic and program flow must, of course, meet the requirements and specifications of the software system being developed. The input and output definition and format requires the interaction of designers, end users, and programmers. In the scheme of system development, software developers need to show the end user how their input/output may look in the finished system. This is an iterative process and may require many prototype input/output formats to establish the proper display interface. This process is very important because the interface with the system can determine the success or failure of a system.

The input interface is important because it is the first

place data or requests that have been entered can be checked for errors. If the input format is complicated or confusing, the end user will make additional mistakes and become frustrated with the system. In this way, the overall effectiveness of the system will be reduced, and the likelihood of erroneous data being entered into the system increased.

The ouput interface is also very important because the end user must interpret what the system has presented to him. If the output format is complicated or confusing, the end user may interpret the output incorrectly.

As I mentioned earlier, the internal logic is, in most cases, left to the designers and programmers to establish. The end user has set forth the requirements and specifications of the system. The internal logic to meet those specifications is determined by the software developers.

"Screen design" tools offer a way in which a designer or programmer can prototype the screen format very quickly. Working with the end user, they can review many different screen formats to insure that the man-machine interface is correctly and accurately designed. This can occur long before any code is generated and helps to reduce the number and complexity of changes once the system is developed. It also encourages the user to take greater responsibility for proper man-machine interface design by showing them the

terminal interactions. Screen design tools can be used for both input and output depending on the type and volume of data to be requested or displayed. The detailed aspects of some screen design software products are described later in this report.

"Report generators" can provide a facility in which the end user is shown different types of reports of data from the system being developed. The software developers can prototype various reports to help the end user determine exactly what types of reports may be needed. This will help insure that the final reports established will meet the needs of the end user and not be complicated or confusing. This effort will improve the overall quality of the system and reduce the complexity and number of changes that may be required to the report section of the software system. When the reports are established they can be coded in the language of the software system. In some cases the software used for prototyping may be adequate enough to be included in the system being developed.

"Application generators" can provide the basis for sound programming. A complex system that works is often derived from a simple system that works. As personnel costs continue to rise, more and more companies try to fill the productivity gap by using application generators. A number of software companies offer these software productivity aids, essentially power tools for programmers. Application

generators use program skeletons that include standard structures for such things as input/output, data definition, and control logic. They include a set of variables that are used to customize the code to specific applications. These program or command procedure skeletons have been tested and are nearly error free. They are a good starting point for many frequently used structures in programming. A significant improvement in productivity can be achieved because these skeletons eliminate much of the design and programming associated with starting from scratch.

Some forms of application generators include the necessary procedures for documenting the programs being developed. They provide program flowcharts. These are developed using the program code as input to a flow charting program. This program developes a logic flow diagram from the program code. These same tools provide cross-reference listings of program variables and subroutines. Some documentation programs read the comments and generate a pseudo document of each module.

Testing

The final phase of software development is testing. Each function of the developed system must meet the specifications and requirements established by the end user. The system is then tested for interaction of all the functions that make up the complete system.

"Software drivers" may be used to test the system.

These software drivers are script files which represent typical requests of the system. These files can be used to stress the system by varying the volume of requests and the extreme values in expected data. These tests can provide a systematic and standardized method of software testing. In this way, any inaccuracies in system results will be brought out.

"Execution analyzers", on the other hand, are used to test the execution characteristics of an entire system or specific parts of a system. First, performance testing involves testing whether the performance criteria specified for the system have been met. The performance criteria for the system may be tested by running a controlled environment in which the load provided by the scripts is representative of the planned workload. For example, the system may be required to process 200 transactions per hour with a mean response time of 2 seconds. The scripts may be set up to provide this workload. Second, execution analysis of program code will show where time is being consumed in the system. From this information, one can determine where code may need to be rewritten or redesigned to improve the overall system efficiency.

Productivity Aids for Screen Design

Most software systems being developed today are interactive, online computer systems. The display terminal is the main interface with such systems. The format of the information displayed on the terminal plays a significant part in the success of the system. It must be well structured and easily understood. The terminal interface is also the first place erroneous data can be screened from the system.

Display Terminal Interface

The display terminal can be used as an interface with the system in either line mode or full screen mode. In line mode the exchange of information with the software system is restricted to one line at a time. This method is relatively inefficient in that only a small amount of information is exchanged with each computer system interaction. In addition, it is difficult to structure requests for data that relate to one another. For example, if you were building a personnel file, you would want to include each persons name, address, phone number, number of dependents, and social security number. In line mode each prompt would request some part of this information. The interactions would be less efficient than full screen mode but each piece of information could be checked as it is entered giving the programmer more immediate control. In line mode the

programmer would also have to be concerned about page scrolling if they wanted to maintain related information on the screen at the same time. Instead of prompting for this information one piece at a time, full screen mode enables the programmer to establish fields on the screen for this information. The user then fills in the fields with this information. After completing or filling all the fields, the information could then be transmitted to the computer for processing. This would greatly improve the efficiency of the interaction. It would also reduce the likelihood of errors because all related information can be seen at one time. Each screen contains the fields or requests for related information and the programmer need not be concerned with page scrolling. Full screen mode tales full advantage of the technology available with most display terminals. It enhances the structuring of the interaction with the computer system. Furthermore, it improves error checking by enabling the programmer or designer to set characteristics for fields on the screen. This capability is really an extension of the programmers error checking routines. In line mode the programmer had more immediate and tighter control in that each field could be checked as they were entered. But in line mode all error checking had to be done by the programmer. In full screen mode some of the standard type of error checking can be done by setting field characteristics and ranges. The programmer can concentrate

more on specifics relating to the data and still maintain control of the data entered. This all adds to improved productivity of the system development effort.

The programming of display terminals for full screen mode, on the other hand, is difficult. In line mode the programmer needs to be concerned with the amount of information that fits on one line and one page. All information is entered into the system one line at a time and error checking is done by the programmer. In full screen mode special programming is required to establish fields on a display screen and to set field characteristics. Furthermore, all the information on a screen is entered prior to transmitting to an application program.

The implementation of full screen mode is dependent on the type of equipment that is used. The products discussed below relate to two types of equipment: the Digital Equipment Corporation VT100 type terminal and its interface with a DEC main frame, and IBM 327x type terminal and its interface with IBM main frame.

Although the purpose of this paper is not to discuss the hardware implementation of full screen, the following discussion should provide enough information to understand how the screen design productivity aids interface with the terminal and main frame.

The DEC VT100 interface with the main frame is at a character interrupt level. Each character is sent and

processed by the computer as it is typed. On one hand this means the man-machine interaction is inefficient in that little information is being processed with each computer interaction. On the other hand, the computer can check each and every character as it is typed and send an immediate response to erroneous information and have it corrected. Then the information that is finally processed by our application program will be error free. The implementation of full screen mode in this environment involves a screen image driver that is the interface between the terminal and the application program. This will be explained in greater detail later.

The implementation of full screen mode for the IBM 327x type terminal is based upon the presence of a mapped character buffer in the terminal, and a terminal controller. There is a fixed one-to-one relationship between each character storage location in the buffer and each character position on the display. For example, if the terminal has a display surface of 24 rows and 80 columns, row 1 maps to the first 80 character storage positions in the character buffer, row 2 maps to the second 80 character storage positions, and so on. All data is entered into the storage buffer of the terminal and there is no interaction with the computer until the SEND key is depressed. The amount of information transmitted is higher than that of a character interrupt system. This improves the efficiency of the

interaction with the computer. The error checking however is not as immediate as with a character level interrupt system.

In either case, the addressability of each character position enables the programmer to establish fields within the display and to control the attributes of each field. Field attributes may include highlighting, color, alpha, numeric, alphanumeric, protected (user can read but not update that field), and field validation (certain items must be present before the display data can be fully processed by the computer. Typically this would be used to insure that a valid processing request was made). The ability to set and control these field characteristics will greatly improve this man-machine interface.

There are a number of other characteristics relating to display terminals that need consideration by the programmers. Special control characters are required to set these field characteristics and control cursor movement. Without some type of programming aid the programmer will have to develop his own low-level language interface. This increases the time needed to develop a screen display and increases the likelihood of program errors.

Terminal Dialog

Numerous vendors offer productivity aids for the programming of display terminals. The products offered contain special functions for the development and use of

interactive applications called "dialogs". A dialog is the interaction of a terminal operator and an application program. The operator communicates his needs to the application program by responding to inquiries presented on the terminal display. Productivity aids, for display terminal programming, are generally grouped into two areas. First is a facility that aids in the development of various types of dialogs. Second, a dialog manager provides control and services to support processing of these dialogs.

"Dialog development" functions increase programmer productivity by simplifying frequently performed programming tasks. They are especially helpful in the development of screen formats. Significant features include:

Full-screen context editing - gives the programmer the ability to design the screen image directly onto the screen. In most cases this is done using a standard editor. After the screen image is created special functions are used to save the image in a screen file for later use.

Screen skeletons and program models - help the system designers and programmers develop dialog panels (screens), messages, function routines, and tables. Theses skeletons and models give the programmer a starting point from which to work. Typically they include the structure of the most likely used areas of a dialog.

Scrolling - enables the user to scroll the information on the screen up, down, left, or right by various numbers of lines or characters.

Interface to standard language processors - provides the structure for linking to and calling other languages.

Dialog test facilities - enables the designer or programmer to display the screen images and trace the flow of information on individual screens and dialog segments.

Documentation preparation assistance - provides a way of structuring the documentation process to improve the documentation of the system being developed. This is done by keeping tables of variables used and processing sequence of the screens displayed during a particular dialog.

Online help and tutorial - these functions help the designer or programmer use the facilities of dialog development. It also enables the programmer to create similar help information and tutorials for the end user of the software system being developed.

"Dialog management" involves a number of functions that provide a variety of services and manages the dialog process. When a dialog is invoked, dialog management:

Displays a hierarchy of screen menus based on user selections.

Invokes functions from the menus such as command

procedures or application programs or displays other screens.

Communicates with the user via data entry displays and messages. This is done through variables on input and messages on output.

Provides online help and tutorial information. The user can reqeust help on using the dialog facility or can request help on the system they are using. The latter help or tutorial information would have to be provided by the system programmer when the dialog was developed.

Generates logs of the interactive session. This facility enables the user to trace an entire dialog, recording all the interactions that occurred.

Maintains user-entered or program-generated data. Dialog management will maintain a list of the last used screens and selections. Then on request, the user can be returned to the last point of the dialog before the dialog was terminated.

Elements of a Dialog

A dialog is made up of a panel, functions, messages, variables, and in some cases tables and file skeletons.

A "panel" is a predefined display image. It may be a menu from which the user selects options, a data entry display, a table display of selected entries from a file, or an information display such as a tutorial or help panel.

A "function" is a program or command procedure that

performs processing or controls the flow of the dialog. The program may be written in an assembler language or high-level language such as FORTRAN; or it may be written as a command procedure.

A "message" is a comment that provides specific information to the user. It may be acknowledgement that a request was received or a warning that something is not progressing as expected.

A "variable" is a character string referred to by symbolic name. It is the main means of communicating between the parts of a dialog such as panels and functions.

Formatting of Displays

The services of dialog development and dialog management functions make it easy for a designer or programmer to define screen display images. These images are specified by panel definitions that include a picture of what the end user will see. This simplifies panel creation and maintenance. Panel and message definitions are created and maintained by editing directly into the panel and message files. Compile or preprocessing steps are not required.

Panel definitions may contain an attribute section, the main body, an initialization section, and a processing section. The attribute section defines the special characters that will be used in the body of the panel to define the attributes of the fields in the panel. The main body of a panel is required and defines the format of the

panel as seen by the user. It also defines the names of any variable fields used on the panel. The initialization section specifies initial processing and typically defines the initial values of variables. The processing section specifies the processing that is to occur after the panel has been displayed. This section defines how variables are to be verified and the functions or other panels that are to be displayed as a result of the values entered on the panel.

The next section reviews a number of products that are designed to improve the productivity of software designers and programmers in their use of display terminals for software development. These products provide special functions for the development and management of terminal dialogs. They are also used by the end user as the interface between the terminal and the application program.

These products represent three levels of capability in products that are currently available. The DEC/Forms Management System provides functions for terminal dialog between the display terminal and application program(s). IBM's Interartive System Productivity Facility provides similar capabilities but provides more system wide services. And finally the SAS/Full Screen Product provides functions for terminal dialog but only as those dialogs interface with the SAS system. These will be explained in more detail below.

DEC/Forms Management System

FMS is a software tool for developing terminal dialogs. These dialogs include panels (screen image forms) and application programs which use the panels for input, output, and control information. FMS panels must be displayed on a VT100 or compatible type terminals. FMS is designed to run on VAX/VMS V2.0.

FMS has two main functions for dialog development:

Form Editor (FED) - is used to design the form directly on the screen. By using FED editing functions you can arrange the fields of the form (context editing), assign form names and field names, and set field attributes that will be used when the form is displayed.

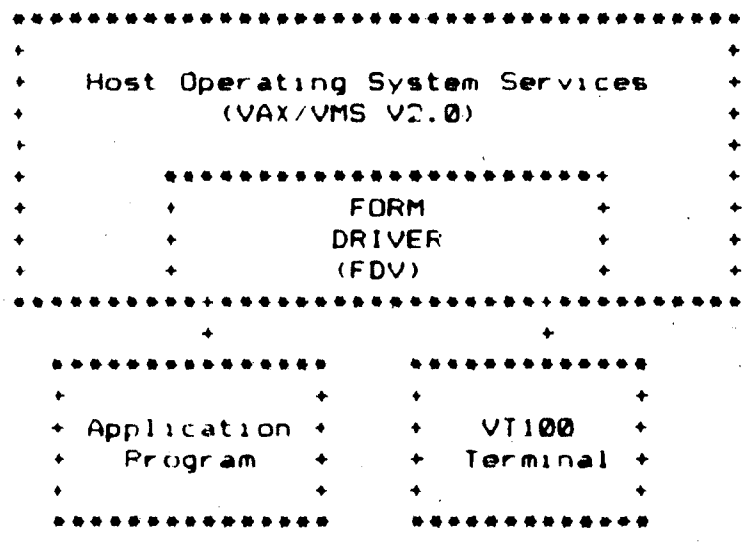
Form Utility (FUT) - aids in the creation of hard copy listings from the forms designed on the screen. It is also used to manage the library of forms already created.

FMS has one main function for dialog management.

Form Driver (FDV) - is the interface between the user's application program and the terminal. The services of FDV are requested by an application program. These services display forms and perform field and character validation from input entered on the form. Once called

to display a panel (screen image), FDV does not return to the calling program until all required data on that panel is entered.

The relationship of FDV and an application program is as follows:



The form driver interface (FDV) is logically between the terminal and the application program. Each key stroke at the terminal is examined by FDV based on established field characteristics. Any errors are immediately reflected back to the terminal user for correction. Once the input for that panel is terminated the data entered is returned to the application program.

Since FDV is the only direct interface with the application program, form maintenance is simplified. Form layout can be changed without recompiling an application program.

DEC/FMS Dialog Development

Creating or modifying forms with the form editor (FED) is an iterative process. The form editor permits one to arrange the fields on the screen and to then test the form to see its appearance on the screen. In this manner, a software designer and user can work together to lay out the form. This will improve the final screen layout and improve the information that needs to be displayed. Programmers can use the editor to display forms in checking the interface between the form and application program. This will improve software development, as a whole, by improving the accuracy and quality of screen image displays. In addition, both tasks of screen design and application programming can continue simultaneously. The only interface is knowing the number and names of variables. Programming for the display terminal becomes a separable task.

The "form editor" has a number of functions to create the screen image and assign field attributes. The fields of the screen image are arranged using the editor. Field pictures are established using a set of picture validation characters which have the following meaning:

C	Alphanumeric
A	Alphabetic
9	Numeric
N	Signed Numeric
X	Any Character

An example of how a screen image might look follows:

```
---------- Employee Records ----------

Employee Serial: 9999999

Type of Changes: AAAAAA(New,Update,Delete)

Employee Name:
     Last       :AAAAAAAAAAAAAAAAAA
     First      :AAAAAAAAAAAAAAAAAA
     Initial    :A

Home Address :
     P.O. Box   :99999
     Street     :CCCCCCCCCCCCCCCCCCCCCCCC
     City       :AAAAAAAAAAAAAAAAAAAAAAA
     State      :AA
     Zip Code   :99999

Home Phone    :
     Area Code:999
     Number     :999-9999
```

The characters in the fields have the meaning described above. This screen image would be displayed without the field picture characters. Any attempt to enter a character that does not match what was specified for that field would be immediately flagged as an error. The terminal operator would have to correct it before he could move on to the next field.

There are additional attributes that can be assigned to each field. The VIDEO function is used to assign video attributes to each field. Possible selections are BOLD, BLINK, REVERSE, or UNDERLINE. Finally, the ASSIGN function displays two questionnaires. One questionnaire is used to assign attributes to the entire screen. The other questionnaire assigns additional attributes to particular

fields. An example of each follows:

```
        Form Wide Attributes

      Form Name       :
      Help Form Name  :
      Reverse Screen  (Y,N)    N
      Current Screen  (Y,N)    N
      Wide Screen     (Y,N)    N
      Starting Line   (1,23)   1
      Ending Line     (1,23)   23

      Impure Area     ???? bytes
      Form Size       ???? words


        Field Attributes

     Name     :
       Right Just(Y,N)_ Clear Char(chr)
     Zero Fill(Y,N)
     Default  :
     Help     :Type a 1, 2, or 3
     Auto Tab (Y,N)N Resp Reqd(Y,N)Y
     Must Fill(Y,N)N Fixed Dec(Y,N)N
     Indexed(N,H,V)N Disp Only(Y,N)Y
     Echo Off (Y,N)N Supv Only(Y,N)N
```

After the form has been designed and attributes assigned the form is saved.

The form utility (FUT) processes the form created by the form editor. It stores the form description in a format that is used by the form driver. FUT can be used to merge forms eliminating the need to recreate a form with the editor. It can be used to print field descriptions and screen images of forms maintained in a forms library. The field description lists each field and the attributes assigned to it. The printed screen image contains a picture of the fields, as they will appear on the terminal, and a

map of the video attributes of each field.

DEC/EMS Dialog Management

The form driver (FDV) processes the form description to display the screen form and collect the responses entered by the user. Using the form description and field attributes, the form driver guides the user through the form. It collects and validates all input based on the form description. Only after all required data is input and validated will the form driver return these values to the calling application program.

The form driver supports interfaces to VAX-11 BASIC, COBOL, FORTRAN, and PL/1. The interface is a call to the FDV driver functions. For example:

CALL FDV$CLRSH(fname,line) - will clear the screen and display the form specified by fname on the line specifed by line.

CALL FDV$GET(fval,term,fid) - gets the value fval, field terminator term, and the field name fid. This is the way in which data is communicated to the application program.

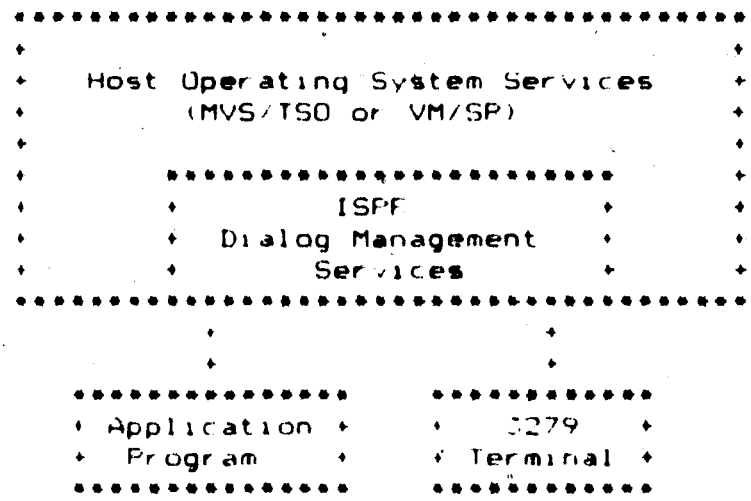
There are other functions similar to those shown above. They are used to accomplish a variety of tasks required to communicate information between an application program and a terminal. These standard functions improve the productivity of programmers by eliminating the need to write their own functions. In addition, the automatic error checking

accomplished through field attributes reduces the chance of error and improves the overall quality of the software system developed.

IBM/Interactive System Productivity Facility

ISPF is an extension of MVS/TSO or VM/SF operating systems. There are two major subsystems to ISPF. The first is "program development" and is an environment in which the programmer can do most all the functions they might do in developing software. It provides a menu interface to all available operating system services such as language processors, compilers, word processors, job submission, electronic mail, editors, HELP facilities, and online tutorials. The second major subsystem is "dialog development anddialog management". ISPF uses its own dialog management facilities to display the panels related to program development. These are also available to designers and programmers for the development of terminal dialogs. ISPF provides the screen management services and display driver interface between the terminal and the main frame.

The relationship of ISPF and an application program is shown below:



In a manner similar to that of DEC/FMS, ISPF dialog management is logically between the terminal and the application program. The information entered on a panel is not reflected to ISPF dialog management immediately but instead stored in the terminal buffer. When the user ends panel input the contents of the terminal buffer are transmitted to ISPF. Each field is examined based on the field characteristics established when the panel was developed. If any errors are detected the entire panel and field contents are redisplayed and the erroneous field marked for correction. Only after all detected errors have been corrected can the terminal dialog continue. Data entered is then passed to the application program or command procedure.

ISPF Dialog Development

Creating a panel (screen image) in ISPF is done by using a standard file editor. Panel definitions include the following sections:

The "body" defines the format of the panel or screen image as seen by the user. This is the picture part of a screen and is required.

The "attribute" section defines special characters that will be used in the body to define field characteristics. It may optionally override default attribute definitions.

The "initialization" section specifies initial processing prior to displaying the panel. Typically, this section defines how any variables are to be initialized.

The "processing" section specifies the processing that is to occur after the screen has been displayed.

Following is an example of a panel as it would appear prior to being displayed by ISPF:

```
%---------- EMPLOYEE RECORDS ----------

%EMPLOYEE SERIAL: &EMPSER
+  TYPE OF CHANGE%===>_TYPECHG + (NEW,UPDATE, OR
DELETE)

+  EMPLOYEE NAME:
+    LAST    %===>_LNAME     +
+    FIRST   %===>_FNAME     +
+    INITIAL%===>_I+

+  HOME ADDRESS:
+    LINE 1 %===>_ADDR1                +
+    LINE 2 %===>_ADDR2                +
+    LINE 3 %===>_ADDR3                +
+    LINE 4 %===>_ADDR4                +

+  HOME PHONE:
+    AREA CODE    %===>_PHA+
+    LOCAL NUMBER%===>_PHNUM     +

)INIT
  IF (&PHA = )
     &PHA = 301
  &TYPECHG = TRANS (&TYPECHG N,NEW U,UPDATE D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  VER (&TYPECHG,LIST,N,U,D,MSG=EMPX210)
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,NNN-NNNN)

)END
```

The +, %, _ signs have the following meaning when processed by the ISPF display service routine.

% (percent sign)	- text (protected) field, high intensity
+ (plus sign)	- text (protected) field, low intensity
_ (underscore)	- input (unprotected) field, high intensity

The trailing plus sign indicates the maximum length of the field. Any attempt to type beyond this field mark will not register and simply tab to the next field.

These are the default settings for field attributes. They may be further defined by including an attribute section. In the attribute section, fields may be declared by:

TYPE(input,output,text)

INTENS(high,low,non)

CAPS(on,off)

JUST(left,right,asis)

PAD(pad character).

The VER statement is verification of the values entered and has a variety of options. The options include:

NONBLANK	- field is required.
ALPHA	alpha only.
NUM	number only.
PICT,string	- matching specific character strings.
RANGE,lower,upper	the value must fall within the limits.
LIST,value1,value2....	the value must be one of those listed.

Although not shown in the above example, the PROC section has another option: %SEL. This option processes the values entered and can select: another screen image for display, a host system command for execution, or another application program for execution.

After the screen is processed by ISPF dialog management the screen would be displayed like this:

```
---------- EMPLOYEE RECORDS ----------

 EMPLOYEE SERIAL:
   TYPE OF CHANGE ===                    (NEW,UPDATE, OR
DELETE)

   EMPLOYEE NAME:
     LAST     ===
     FIRST    ===
     INITIAL  ===

   HOME ADDRESS:
     LINE 1   ===
     LINE 2   ===
     LINE 3   ===
     LINE 4   ===

   HOME PHONE:
     AREA CODE     ===
     LOCAL NUMBER  ===
```

ISPF Dialog Management

ISPF dialog management services can be executed from a command procedure or high level language such as FORTRAN, PL 1, or COBOL. It provides a number of services for dialog management.

"Display Services"

DISPLAY - reads screen image definitions from screen files, initializes variable information from variable files, and displays the screen on the display terminal. After the data is entered on the screen, the inputs are stored in dialog variables and the display service returns to the application program or function.

TBDISPL - combines information from screen definition files and ISPF tables. It displays selected rows and

columns and permits the user to select the rows for processing.

"Variable Services"

Dialog variables serve as the main communication between dialog functions (application programs or system commands) and ISPF services. Variable services allow a function to define and use variables, referencing them symbolically by name.

"File Tailoring Services"

File tailoring services read skeleton files and create current tailored output that may be used to drive other functions. Typically, these services are used to modify a job processing step by filling in variable values for a particular job. In addition to this, they can be used to help standardize parts of software development such as documentation. A skeleton can be as simple as providing a standard way of recording information about a program module. This may be module name, programmer name, creation date, and the date the module was tested or modified. For example:

```
        MODULE - &MODNAME
    PROGRAMMER - &PRGNAME
       CREATED - &DATE
        TESTED - &TESTDATE
      MODIFIED   &MODDATE
```

The skeleton stub above could be included in other screen images and dialogs to insure that this specific type of information is recorded. It is general enough,

however, to be included in a variety of documentation steps. In this way, a library of standard skeletons can be developed to record necessary information.

File tailoring services are provided by four functions:

FTOPEN - prepares the file tailoring process.

FTINCL - specifies the skeleton to be used and starts the tailoring process.

FTCLOSE ends the tailoring process.

FTERASE - erases any output file that was created by file tailoring.

ISPF can be invoked in two ways: from a command procedure or from a programming language. Languages that are supported are FORTRAN, PL/1, and COBOL. ISPF services are invoked in the following manner (or form):

from a command procedure

```
ISPEXEC service-name keyword1(value) keyword2(value)
ISPEXEC DISPLAY PANEL(screen image name)
```

from a program

```
CALL ISPLINK (service-name, keyword1(value),
                    keyword2(value) .....)
CALL ISPLINK ( DISPLAY , PANEL(screen image name) )
```

ISPF is very similar to DEC/FMS except for the actual hardware implementation as discussed earlier. They both provide flexible yet standard ways to develop and implement

terminal dialogs. They both have functions that provide for: panel development and display, panel interface to standard languages, and dialog test facilities. The software implementation of panels, fields, field characteristics, and the control of dialogs is different but only in the actual instructions used, conceptually they are the same.

Statistical Analysis System/Full-Screen-Product

The Statistical Analysis System (SAS) is a set of procedures that can be used for all types of data creation, modification, analysis, and retrieval. SAS/Full Screen Product (FSP) runs within the SAS environment, which runs under the IBM operating systems MVS/TSO or VM/CMS, using an IBM 327x or compatible display terminal.

This product represents a set of products similar to the two previouly discussed. It is very limited in scope though in that it can only be used within the SAS interactive subsystem to create, manipulate, or display SAS datasets. Although its application to computing is restricted, it may suffice in particular data processing situations.

SAS/FSP has one procedure FSEDIT for developing terminal dialogs. FSEDIT accomplishes both functions of dialog development and dialog management in a limited fashion. Full terminal dialog as previously discussed is not possible with SAS/FSP. SAS/FSP has three additional procedures but they are not used for developing and managing terminal dialogs. The four procedures are:

PROC FSEDIT is used to change panel (screen image) layouts and manipulate SAS datasets.

PROC FSBROWSE is used to display observations in a SAS data set.

PROC FSLETTER is used to edit and send letters through

electronic mail.

PROC FLIST is used to list SAS datasets.

These procedures provide the designer, programmer, or end user with the ability to work with their SAS files. Since SAS is already an interactive subsystem, the end user is typically the person using SAS/FSP.

A SAS data set has two parts: a descriptor section containing documentation about the data and a data section containing the data values arranged in a rectangular table. Rows of the table represent observations; the columns represent variables which are identified by name. The descriptor section stores information about each variable such as: type (character or numeric), length, position in the table (column), format (format for display or print), informat (format on input), and a label (description for a variable name). Historical information may also be contained in the descriptor section stating when the data set was created, last update or modification, and the statements used to create the data set. SAS data sets are the basis for the full screen procedure FSEDIT.

FSEDIT is a procedure that is used to add, delete, update, or locate observations within data sets. It is also the procedure that is used to layout the fields on a screen panel and define any special characteristics of the fields. On execution of FSEDIT a default layout of fields is used if no specific screen layout was requested.

As an example, assume that you wanted to keep a mailing list of subscribers to a magazine. Using SAS you would create the description section of the data you wanted to keep as follows:

```
DATA IN.SUBSCRIB;
  INPUT name $50. #2 ADDR1 $50. #3 ADDR2 $50.
    #4 BEGDATE MONYY5. #5 YRS 1. #6 NEW $1. ;
 INFORMAT BEGDATE MONYY5. ;
 FORMAT BEGDATE MONYY5. ;
 LABEL BEGDATE=DATE OF FIRST ISSUE
    NEW= 1=NEW SUBSCRIPTION, 0=OLD SUBSCRIPTION ;
 CARDS;
```

This represents a DATA step within SAS. As a result of executing this code SAS creates a data set with the following descriptor section:

#	VARIABLE	TYPE	LENGTH	POSITION	FORMAT	INFORMAT
		LABEL				
1	NAME	CHAR	8	4		
2	ADDR1	CHAR	50	54		
3	ADDR2	CHAR	50	104		
4	BEGDATE	NUM	8	154	MONYY5.	MONYY
		DATE OF FIRST ISSUE				
5	YRS	NUM	8	162		
6	NEW	CHAR	1	170		
		1=NEW SUBSCRIPTION, 0=OLD SUBSCRIPTION				

To display the screen image you invole FSEDIT in the following way:

```
PROC FSEDIT DATA=IN.SUBSCRIB;IN.SCREEN1;RUN;
```

This command is executed from within the SAS environment and specifies the data set to be processed. The screen file used to save the format and variable characteristics is also specified. The first screen displayed is the primary option menu screen.

FSEDIT Primary Option Menu

Select option===> Press END to return to SAS

1 Edit SAS data set: IN.SUBSCRIB

2 Review or change PF key definitions

3 Review edit commands

4 Review the PROC FSEDIT statement and options

Screen Modification

5 Review or change screen modification PF keys

6 Modify the edit screen

To arrange the format and set characteristics of the fields specified in the data set you select option 6. A screen is presented with the variables displayed in a default format.

EDIT SAS data set: IN.SUBSCRIB

Command ===>

NAME:
ADDR1:
ADDR2:
BEGDATE:
FRS:
NEW:

A variety of facilities are available for screen design and the setting of field attributes. Using the editor, fields can be arranged in whatever format the designer or programmer desires. After the format is determined, the attributes can be set for each variable or allowed to default to system standards.

The variable field attributes are set from special screens selectable by command names. Each time one of these commands is issued a screen image of the fields defined is displayed. These special screens set field attributes as follows:

INITIAL - specifying a value in a field sets its initial value. These will be displayed as initial values each time the variable is displayed.

MAX MIN - by placing a number in a field sets a range of acceptable values. If a value is entered that does not fall within these values, an error messages is displayed.

REQUIRED - an R in any field specifies that this field must be entered before the screen can be processed.

CAP - a C in any field specifies that this field will be capitalized when processed. Fields not marked in this way, will remain as they are entered.

COLOR - placing the first letter of a color in a field sets the color of this field when it is displayed.

ECOLOR - one can specify the color of a variable field when an error has been detected. The color is specified in the same manner as for COLOR.

ATTR - attributes of each field can be set to one of a number of attributes such as: blinking, reverse video, or underline.

PROTECTED - this screen lets you define protected fields

by entering a P in any field you want protected. Protected fields appear on the screen but can not be altered.

Once the format and characteristics of the variable fields are set they are saved for later use. At any time the designer or programmer wishes to change the screen they use FSEDIT and specify the data set and screen they want to change.

Although similar in concept to the previous two products, SAS/FSP is limited to the SAS environment. This reduces its value as a general facility for creating and managing terminal dialogs.

CONCLUSION

The three dialog management products described are typical of the products currently available. Some, like SAS/FSP, are very specific to one type of environment. Others, like DEC/FMS and IBM/ISPF, have a wider application to interfacing with a host system or application program. All offer facilities to easily create and manage the interface with the computer.

Each product is designed to run on specific hardware. However, as software engineering continues to improve programming techinques, products will evolve that will have application across different types of hardware and operating systems.

The computing industry has been driven by advances in hardware technology. But, as it grows, the pressure appears to be for software to become the driving force behind effective growth of its consumers; their time is now the most expensive item. Actions to reduce the human costs and simplify the man-machine interface will have the greatest impact on these costs. Office automation facilities, design and coding tools, application and report generators, and program test routines meet some of these needs. They provide special routines that standardize frequently used functions and make them easier to use. Screen image design and dialog management provides the tools for development of effective display terminal interfaces.

Merged into one coordinated set of tools, these productivity aids will improve the quality of software systems and the productivity of the software development effort.

BIBLIOGRAPHY

Arthur, L. J., "Programmer Productivity", John Wiley and Sons, New York, 1983.

Bildson, R., "Programmer Productivity: Plenty of Productivity Aids; No Guarantees", Software News, Vol. 3 No. 10, October 1983.

Booch, G., "Software Engineering with Ada", Benjamin Cummings Publishing Co., Menlo Park, Ca., 1983.

Booze, Allen, and Hamilton, Inc., "Booze, Allen Study of Managerial/Professional Productivity", Booze, Allen, and Hamilton, New York, 1980.

Bracket, M., "Programmer Productivity: What's our current situation?", Software News, Vol. 3 No. 10, October 1983.

Card, S. K., T. P. Moran, A. Newell, "The Keystroke Level Model for User Performance Time with Interactive Systems", Communications of the ACM, 1980.

Dean, M., "How A Computer Should Talk To People", IBM Systems Journal, Vol. 21 No. 4, 1982.

Doherty, W. J., "The Commercial Significance of Man Machine Interaction", Infotech State of the Art Conference, November 1978.

Farber, R. M., "Programmer Productivity: Benefits and Pitfalls of Generators", Software News, Vol. 3 No. 10, October 1983.

Leavitt, D., "Programmer Productivity: Measure Programs, Not Programmers", Software News, Vol. 3 No. 10, October 1983.

Lent, R. M., T. L. Booth, T. T. Wetmore IV, "An Instrumentation System For Measurement of Software Performance", Proceedings Computer Software and Application Conference, November 1982.

Lodding, K. N., "Iconic Interfacing", IEEE Computer Graphics and Application, March/April 1983.

Mandell, S. L., "Computers and Data Processing Today", West Publishing Co., St. Paul, Mn., 1983.

Putnam, L. H., D. T. Putnam, L. P. Thayer, "Programmer Productivity: Software Equation Computes Characteristics", Software News, Vol. 3 No. 10, October 1983.

Stevens, A., B. Roberts, and L. Stead, "The Use of a Sophisticated Graphics Interface in Computer-Assisted Instruction", IEEE Computer Graphics and Application, March/April 1983.

Weizenbaum, J., "Computer Power and Human Reason", W. H. Freeman and Co., 1976.

Wozny, M. J., "The Human-Machine Connection", IEEE Computer Graphics and Application, March/April 1983.

Digital Equipment Corp., "VAX-11 Forms Management System", Software Reference Manual No. AA-J269A-TE, 1980.

IBM Corp., "Interactive System Productivity Facility", Program Number 5668-009, March 1981.

SAS Institute, "SAS/Full Screen Product", Users Guide, 1982.

CANDIDATE BIOGRAPHY

Lester L. Zern is a senior programmer analyst at General Electric in the Space Systems Division. Consultant to a number of data processing complexes, he advises in the areas of operating system tuning and productivity. Before his employment at General Electric he was an assistant research engineer at Bethlehem Steel Corporations Homer Research Laboratories. At Homer his responsibilities included the computer operating system software and the laboratories data communication interfaces with the computer center.

A 1971 graduate of Capitol Radio Engineering Institute in Washington, D.C., in Electronic Engineering Technology majoring in Communications Engineering Technology. He then attended Ursinus College in Collegeville, Pa. prior to entering Lehigh University.