EDP ANALYZER

OCTOBER, 1979
VOL. 17, NO. 10

## PROGRAMMING WORK-STATIONS

It seems that many data processing departments are very much like the proverbial shoemaker whose children go barefoot. These departments spend a lot of time developing sophisticated tools for other departments in their organizations—and forget about their own staffs' needs. We now see this changing dramatically, with the introduction of programming work-stations. In this and next month's reports, we explore these exciting new products and systems for software development.

Joshua Tree Manufacturing, Inc. is a privately held manufacturer of junior-sized women's sportswear. They have 500 employees and regional showrooms in five major U.S. cities. Their headquarters are in Redondo Beach, California, a suburb of Los Angeles.

Data processing at Joshua Tree is done on an IBM 370/125, to which twenty local terminals are connected for on-line programming and maintenance as well as for order entry, credit management, accounting, etc. The data processing staff consists of a manager, a system programmer, two programmer/analysts, two computer operators, and two data entry people.

In early 1977 the people at Joshua Tree began looking for a software library package to maintain their source programs. Their IBM sales representative suggested that they consider what was then a new IBM product, ETSS (Entry Timesharing System), which would allow on-line programming as well as source program maintenance. Since the data processing manager had previously used and liked an on-line programming facility, this product looked interesting. So he and the system programmer visited a company they knew that had been using ETSS for over a year. These users were very pleased with ETSS because it was a time sharing system aimed specifically at software development. So in 1977 Joshua Tree installed ETSS and gave a CRT work-station to each programmer in the department, as well as to the manager.

The manager told us that ETSS has provided three major benefits for them. First, it has spread out the use of computer resources more evenly over the software development life cycle. Second, it has made project tracking easier and more accurate. And third, it has improved programmer productivity by 60-80%.

Formerly, specific hours were allocated on the 370/125 for software compiling and testing. More often than not, there was either no demand or too much demand for this time. In this environment, programmers tried to pack as much code into each compilation or test run as possible. This obviously accentuated the computer usage peaks.

Using ETSS, the programmers no longer try to code as much as they can before each compilation or test. With work-stations in their offices, they code directly into the system in PL/1, using ETSS macro commands, editing features and files. Being assured of computer time whenever they want it, they now code one module at a time, compile and test that module, and then move onto the next module in their programs. Thus, use of computer resources begins much earlier in the development cycle and is spread out more evenly over the cycle and over each working day.

This new procedure also makes project tracking easier and more accurate, because the development cycle now tends to be broken into shorter segments. Completion of each task is easier to verify. "You do not have to wait until the sixth day of a phase to find out if the three modules in that phase have been written, compiled and tested. You can find out on the second day whether the first module has been finished, on the fourth day whether the second module has been completed, and so on," the manager told us.

ETSS increases programmer productivity in several ways. For one thing, it eliminates a lot of programmer waiting—waiting for computer time, waiting for compilations to finish, etc. Once a program section is coded, the programmer enters it into the compilation and test queue and then goes onto other work. Periodically he can inquire from his work-station on the status of the job. When the job is finished, he can get test run output at the work-station.

Additionally, the command language in ETSS has greatly improved programmer productivity. ETSS provides some twenty standard commands that can be invoked in a program simply by calling the command's name. On an even more sophisticated level, these ETSS commands can be linked to form a macro procedure, given a name, and invoked by calling that name. These macro procedures eliminate a lot of duplicate coding, we were told. The on-line editing features of ETSS make it easy to call up a procedure or a PL/1 module, tailor it by changing some lines, rename it, and use it in a new program. All of the programmers at Joshua Tree have their own libraries of such modules. Some of these they designate as being private, for their own use only, while others are designated as public, for use by others.

After Joshua Tree had been using ETSS for four months, they visited a company that had been using it for much longer. This visit proved most helpful, because they learned how to use the more sophisticated features of ETSS.

The data processing manager at Joshua Tree pointed out that taking advantage of the potential of ETSS required two company policies: (1) giving a work-station to every programmer, and (2) developing on-line programming standards. One such standard is that every updated version of a program must be run through an ETSS module that they wrote. This module creates a new source listing for documentation purposes as well as a new object listing for the operational program library. So documentation reflects the current production version of every program.

At Joshua Tree they are very enthusiastic about their use of IBM's ETSS, and they recently replaced it with ETSS II which has additional capabilities.

## Abacus Systems, Inc.

Abacus Systems is a software and OEM company that specializes in writing business application software for Hewlett-Packard systems. It then sells this software to end users and other OEMs. Abacus Systems is located in San Francisco, California.

Abacus Systems was one of the first companies to install an H-P 300 system, early this year. The H-P 300 is a small business system with some very interesting software development tools built into it. It has two types of work-stations connected to it. One is the major controlling terminal called IDS (Integrated Display System), which can be used as a stand-alone program development work-station. The second type is the H-P 264X terminal; up to 16 of these terminals can be connected to the IDS.. These terminals can be used in a data entry mode, for entering code or on-line testing of a compiled program. They can not be used for editing or compiling programs.

At Abacus their H-P 300 system has the IDS work-station with its built-in floppy disk unit, one 264X terminal, a 180 character per second printer, and a 7906 hard disk unit. This latter

disk unit has both removable and fixed disks, which provides Abacus with a convenient way to dump files for back-up purposes, we were told. They use their system to write application programs in Basic, to run on other H-P 300s.

Use of the system for creating software goes as follows. First, after designing their program modules, the programmers code or partially code their designs on coding sheets. With these in hand they sit down at either type of work-station (the IDS or the 264X) and enter the code. Then from the IDS work-station, they initiate a compilation of one or more modules. Upon completion of the compilation, the system automatically displays how many syntax errors have been found.

To correct an error the programmer pushes the 'softkey' on the IDS screen designated NEXT ERROR and the screen splits into two sections. The top section describes the first compilation error found. The bottom section displays about 10 lines of source code, with the cursor positioned where the system thinks that error is. The programmer corrects the error using the work-station's editing features and then pushes the NEXT ERROR softkey again. This time the system displays the second syntax error. And so debugging goes.

When all of the errors have been corrected, a new compilation is requested. The programmer can also ask the system to extend the test by running the link editor to resolve references between modules, and then run the program using a specific test data file. During this processing, the system allows interactive symbolic debugging by the user.

After using the system for several months, the people at Abacus Systems have speeded up their software development considerably, by re-using previously written modules. For instance, one general ledger program creates a formatted screen display with which an end user can request a number of types of reports with various options. These report requests are then processed and passed on with instructions on how each report should be printed—that is, its format, number of copies to be printed, etc. This program contains some 750 lines of code, in three modules. It was created in about one hour, because the programmer pulled existing modules out of the H-P 300 file, modified them, and then linked them together.

Abacus Systems is creating quite a library of modules that they can use over and over again using the on-line editing features of the H-P 300.

The people at Abacus say that after six months of use they began to get very accomplished at using the more sophisticated features of the H-P 300. They pointed out to us that it is quite a complicated system because of its numerous special features, so it takes some time to begin using it well. But now that they are knowledgeable in its use, it is very dramatically increasing their programmer productivity.

## Buck Knives, Inc.

Buck Knives is a leading manufacturer of high quality folding and sheath knives for hunting and general use. From their headquarters in El Cajon, California, a suburb of San Diego, they distribute their products to both domestic and international markets. Until early last year, Buck Knives had been using a software house to write programs to run on their DEC PDP 11/70, which uses the RSTS time-sharing operating system. In mid-1978 a data processing department was formed at Buck Knives. It consists of a manager, an entry level programmer, and an operations staff.

About one year ago the manager began investigating better ways to develop software, because the systems he had inherited were becoming inadequate; he also wanted to institute an on-line programming environment. In his study he came across USER-11, a software development package developed by North County Computer Services, a service bureau some forty miles from him.

The USER-11 package operates under the RSTS operating system. It includes a data management system and some 25 conversational programming procedures for use with DEC's BASIC PLUS language. Also a text editor is provided.

The programming approach is the most interesting and unique feature of USER-11. To a major extent, it involves *non-procedural* conversational programming. That is, USER-11 provides a number of generalized facilities—for

defining data records, allocating disk storage space, adding, deleting, and changing records, selecting, sorting, and displaying records, and preparing reports. The user specifies *what* a desired new program should do by selecting among the options given by USER-11.

For example, if a report program is to be written, the programmer invokes the REPORT procedure. This procedure asks a series of questions by which the report's contents and formats are defined. The control parameters that are created may be used only once, if desired (since creating them takes only a few minutes), or they may be stored for repeated use.

For writing one's own code, the programmer uses the CREATE procedure. This procedure also utilizes the question and answer approach to create a documented, standardized BASIC PLUS program to which the programmer adds any necessary main line code.

The manager at Buck Knives thought USER-11 looked very flexible, and it would eliminate a lot of coding. So it was purchased in late 1978. After about two weeks of training and use, both the manager and the entry level programmer were quite proficient at using USER-11 for program development.

An example of developing a typical system using USER-11 is as follows. The company controller requested a program for tracking certain general ledger accounts. The program was ascertained to require six modules: (1) data entry posting to a batch file, (2) batch file listing, (3) batch file maintenance, (4) batch posting to a master file, (5) master file maintenance, and (6) master file reporting. So this was more than just creating a new report.

The initial system design took about one-half hour, then 'coding' on-line using USER-11 procedures began. 'Coding' and testing were accomplished in three hours. They consisted of the following eight steps. We found it interesting that in these steps the programmer concentrated on *what* needed to be done, leaving the details of *how* to do it up to the USER-11 procedures.

First, using the ALLOCATE procedure, space for the master and batch files was created.

Second, using the DEFINE procedure, the data definitions for the various fields in these two new files were specified. Since both use identical data, only one data dictionary was actually created.

Third, using the on-line editor, an ASCII text menu control file of all items, security levels and job control commands was defined. When run, this control file requests the end user's identification number and password, and then provides the user with a menu of procedures that he or she is authorized to use.

Fourth, the user identification, password and security level information were entered into the security file.

Fifth, the security and menu portions of the system were tested.

Sixth, modules one, three and five were tested. These three modules, which process posting to a batch file, batch file maintenance and master file maintenance, needed only to be listed on the menu with the operation options to be performed on them specified. USER-11 takes care of the details of data maintenance—a major reason why it requires a lot less coding than conventional procedural programming.

Seventh, the three remaining modules were 'coded' by creating USER-11 EXECUTE files. These ASCII text files are created using the RECORD procedure and are used by the EXECUTE procedure when the program is run. For the batch file listing, module two, the programmer specified that the SORT procedure was to be used to sort all records by specific keys, which he designated. And using the REPORT procedure, he created the report recall file, which contains all of the parameters necessary to generate the report.

Finally, in the eighth step, the entire system was tested.

The manager estimates that using a procedural programming language, this project would have taken 20 hours to design and 60 hours to code and test; instead of the three and one-half hours actually spent using USER-11.

The ninth step, user training, we also found to be interesting. The programmer gave the controller the required user identification and password information and told him, "If you do not understand its use at any point, just type in /HELP and USER-11 will explain your options." All of the USER-11 procedures are fully documented, on-line, so that user training is just

that simple. The menu, created in step 3, gives the valid user the program's options, and the HELP command explains each option.

So the final programs are easy to use. And they are efficient to run, we were told, since the USER-11 procedures are coded in highly efficient code.

Further, these programs are easy to maintain, because maintenance usually consists only of changing options, such as on a report. It does not consist of recoding how the report is to be produced. Therefore, program maintenance has been reduced by almost 80%, the manager speculates.

"At Buck Knives," the manager told us, "USER-11 has really spoiled us. We know that projects that would normally take several months now take only a couple of weeks. Out of 95 production programs in our integrated order entry and accounts receivable system, we only had to partially code (using CREATE) 16 programs. The remaining programs were replaced entirely by USER-11 modules. So we had to code 83% fewer programs, which saved us $16,000 in labor costs alone."

## Programming work-station types

Programming work-stations, as we define them, are those portions of computer systems that have been developed specifically for use by programmers for software development and maintenance. We call these hardware/software systems 'work-stations' because we foresee them becoming the main tools with which programmers will perform their work. As such they are designed to replace: coding sheets, pencils, card decks, paper listings (sometimes), walks to the computer room (to submit and pick up jobs), telephone calls to the computer room (to find out if jobs have been run), project tracking charts, and even documentation typists. Within the past year, we have seen a number of new programming work-stations come on the market (Reference 4). With hardware costs dropping dramatically and personnel costs rising steadily, these products appear more cost effective to data processing management.

We categorize programming work-stations into two main types: (1) host programming work-stations and (2) stand-alone programming systems. Host programming work-stations are software or firmware products that create a programming work-station environment on a company's in-house computer. Computer programs developed using these tools are meant to run on the same computer. We discuss these products in this report. Stand-alone programming systems are total systems designed for software development use only. As such, the computer programs developed on them are (generally) meant to run on another system, called the target system. We discuss these systems next month.

Before starting, we should define a few terms that we will be using, to avoid confusion. *On-line programming* refers to entering code and making changes at a computer terminal or work-station, rather than on cards. The terms *conversational*, *interactive* and *dialog programming* all refer to programming on-line with the system prompting the user. It is a question and answer approach to programming.

Most programming done today is *procedural programming*. That is, programmers use COBOL, BASIC, PL/1, etc. to write down the procedures for *how* to do something. On the other hand, in *non-procedural programming*, programmers need only specify parameters and select options presented by the system. The programmer is simply concerned with specifying *what* needs to be done, not with *how* to do it. The system then creates the procedural code to perform the task. Now, onto our discussion of host programming work-stations.

## Host programming work-stations

Typically, on-line programming has been performed on the same system that the programs are intended to run on. One method of providing programming work-station support on a host computer is to install a software package developed for that purpose. Such software packages have been available for some time. And we have recently seen some impressive new offerings. These packages operate under the host's time-sharing operating system.

Some packages aim to enhance procedural programming, while others aim at enhancing productivity through non-procedural programming. We see non-procedural programming as a move toward end-user programming.

We found software packages such as these to be a very viable way to move into on-line programming. This is especially true for small data processing departments that cannot justify the cost of a stand-alone programming system. But an important criterion is that this programming use not consume a lot of computer resources and degrade the host's performance.

Two examples of software packages for programming work-station support are ICCF (and ETSS) from IBM and USER-11 from North County Computer Services.

### ICCF and ETSS from IBM

Somewhat hidden in IBM's announcement last January of their new 4300 computer mainframe family, to replace System/370, was their introduction of ICCF (Interactive Computing and Control Facility). ICCF is a programming work-station product. It is an enhanced version of ETSS and ETSS II and is designed to run on System/370, 3031 and 4300 computers under the DOS/VSE operating system.

When used with non-intelligent terminals, ICCF provides line editing capabilities, also known as command or context editing. That is, to make a change in a line of text, the programmer types in the specific command for the change to be made followed by the old characters to be changed and then the new characters to be added.

Using an IBM 3270 terminal, full-screen editing is possible. In this mode, the programmer moves the cursor to the position on the CRT screen where a change is to be made. Then he or she pushes the function key that designates the type of change to be made and types in the new characters. This mode of editing is much faster and requires a lot less typing.

Use of the 3270 terminal also allows the programmer to split the 43-line CRT screen into as many as eight windows. This is very handy for comparing two modules or looking at compiler run error messages in one window and source code in another.

ICCF provides users with a work area for entering new work or editing stored programs. All work is transferred to the work area before it can be edited. This protects the originals from accidental destruction.

ICCF also provides users with 'libraries' into which they can store source programs, data, procedure modules, and job control statements. Keeping standard JCL strings in a library relieves programmers of re-entering this information every time they submit a job. (And ICCF helps programmers create these strings in a conversational mode.) These libraries can be designated as private, for a programmer's personal use only, or public, for anyone's viewing and use.

Jobs to be compiled or tested are placed into the host's batch queue by ICCF for execution. A special partition can be reserved on the mainframe for compilations and tests. Testing does not tie up the terminal, so the programmer can go onto other work after submitting a job. And he or she can periodically inquire about the status of the job from the work-station. When the run is complete, results are available for viewing on the work-station.

ICCF supports conventional procedural programming. It also supports non-procedural programming using IBM's DMS/VS development management system, when using the full screen editing capabilities. And it contains a command language of some twenty macro commands. These can be executed by a programmer to perform their functions or they can be strung together in a sequence to form a macro procedure. Both macro commands and macro procedures can be included in conventional programs by simply specifiying the macro name. Users can also create their own conventional modules, give them a name, and use them as macro procedures.

ICCF appears to be a real bargain. It is listed as costing $60 a month plus a monthly support fee. To find the equivalent cost to other work-stations, pertinent other costs—including both purchase and operating costs—need to be added to this figure. For more information on ICCF and ETSS contact your local IBM sales office. You may wish to request the ICCF general information manual (Reference 1).

### USER-11 from NCCS

USER-11 is a data management system that was developed by North County Computer Services in Escondido, California. NCCS offers USER-11 to its time-sharing customers through-

out San Diego County, and it also sells the package. The system has been sold in the U.S., Mexico, Australia, and New Zealand.

USER-11 runs on any DEC PDP-11 computer under the RSTS/E VO6-B operating system (or more recent version) with at least 64k words of memory. It works in conjunction with DEC's BASIC PLUS language and most CRT or printing terminals.

USER-11 is both a development and an operations system. That is, it provides generalized facilities for developing application programs and it performs database management services when the application programs are run. Many application programs can be created using just the generalized non-procedural facilities within USER-11. In those cases where tailored program logic is need, BASIC PLUS can be used with the CREATE procedure.

There are currently 25 procedures available. Some of them are: (1) ADD—to add records to a database, (2) DEFINE—to establish and maintain a data dictionary that is used both by the system and by users, (3) LABEL—to print address labels, (4) LINK—to link several files together using a master file, (5) REPORT—a general purpose report generator, (6) UPDATE—a general purpose, interactive database field update program, (7) CREATE—a BASIC PLUS program generator for which the programmer simply supplies main-line program logic, and (8) EXECUTE—a program generator for including USER-11 procedures in application programs.

To use a procedure, the programmer types: USE XXXX, where XXXX is the name of the procedure. The system identifies the procedure requested and asks the programmer the first question. For example, to add new fields to a file, using DEFINE, the system first asks for the name of the file. The user replies and then the system tells the user how many fields have already been defined in that file. When the user types ADD the system displays the next usable field number and in a dialog fashion it successively asks for: the new field's name, its length, any explanation needed to clarify the meaning of the field, the field's header name to be used on reports, its edit mask, and various field specifications. If at any time during this dialog the user does not understand what the system is asking, he types /HELP. An explanation and a sample response are provided. If, for example, the user does not understand EDIT MASK, the system explains the term: EDIT MASK is to be used when printing the contents of this field. Possible options are $, zero fill, CB (credit balance), - (negative number), and underscoring. The /HELP command can be used when programming with or using USER-11.

USER-11 users told us they rarely write conventional BASIC code any more. They find that the USER-11 procedures fill all of their programming needs, except when complex data entry or link processing is required. And NCCS is currently writing two new procedures, FORM and ENTRY, to handle these situations.

USER-11 currently sells for $9500. For more information, see Reference 2.

A second type of host programming work-station provides programming work-station support through firmware rather than software. So the features can not be purchased separately; they are an integral part of the system. Programs developed on this type of host system are meant to run on it also.

This approach, when used on a small business system, looks very interesting for application development for distributed systems. The programs can be developed (and maintained) centrally, using the system's programming tools, and then distributed to the various business units, for running on the same type of small system. If non-procedural programming capabilities are also available, end users may even be able to develop their own programs.

Our example of this type of programming work-station is the H-P 300, a small business system from Hewlett-Packard.

### H-P 300 from Hewlett-Packard

Hewlett-Packard's 300 computer system is basically a system in a desk. It is aimed at the small business and distributed data processing markets. And it is designed to be dedicated to a few applications at the department level. But it also contains some powerful and unique tools for business software development.

The basic H-P 300 system includes an IDS terminal with its floppy disk storage and keyboard, a 12 million byte fixed disk and a 16-bit processor, using three silicon-on-sapphire chips, with 256k bytes of memory—all built

into the desk cabinet. This basic system costs $36,000. It can be expanded by increasing memory up to 1 million bytes, and by adding up to 16 data entry terminals, two printers, and/or larger disk units with removable packs. It supports H-P's extended version of BASIC, RPG II, and System Language 300.

Of interest to us here are the IDS (Integrated Display System) and the programming language sub-systems.

The IDS is a very powerful display system which allows: split screens, with each window having I/O and editing capabilities; vertical and horizontal scrolling; inverse video (black on white) to highlight particular items on the screen; command-driven and full-screen editing; and softkeys. The softkeys need some explanation.

To the right of the CRT display are eight push-button keys, which Hewlett-Packard calls softkeys. The current meanings of the keys are displayed along side each on the screen. So the functions of these keys can and do change.

Typical softkey functions available during BASIC programming are: (1) HELP—to gain access to the system's on-line quick reference guide, (2) SINGLE SCREEN, (3) SCROLL UP/DOWN, (4) TEST—to automatically compile, link (if several modules are being tested) and execute programs, (5) OOPS!—to restore the last change made back to its original state, (6) EXIT BASIC—to exit the BASIC language sub-system and enter the operating system or another environment, (7a) COMMAND—to move the cursor out of the editing window so the user can enter commands to the system, and (7b) COMPOSE—to move the cursor into the editing window to perform full-screen editing. A different set of softkey functions is provided during debugging.

The H-P 300 has three programming language sub-systems—RPG II, BASIC, and System Language 300. System Language 300 is aimed at software specialists who want to write assembly level code. Each sub-system contains a number of programming aids aimed specifically at that particular language. For BASIC there is a syntax checker that notifies the user of an error immediately after a line of code has been entered. There is an extensive command language of some 70 commands. Some are for direct use, and some require some additional variable information (such as file name) to perform their tasks. The command interpreter has sufficient intelligence so that the user can enter abbreviations and minor mis-spellings which the system will still understand.

These are the most unique programming features of the H-P 300 small business system. For more information on this system, see Reference 3.

## Stand-alone programming systems

Stand-alone programming work-station systems are relatively new offerings. They are full-blown hardware/software systems created specifically to support software development. As such, the programs written using them are (generally) *not* meant to run on them but on different computers, known as target systems. Thus, these systems require an added communication feature for sending programs to the target computer for compiling and testing. The stand-alone systems that we have seen cost anywhere from $70,000 to almost $200,000. We shall discuss them next month.

This brings up the question of money and whether these systems can be cost justified.

## Can they be cost justified?

Management's initial reaction to these programming work-stations is likely to be: "Say, these look great, but they also look expensive. Can they be cost justified?" Well, the users we talked with say Yes, based on the benefits they are receiving.

To help others better quantify possible benefits, to see if these systems can indeed be cost justified, we will now discuss the claimed benefits. Unfortunately, we have seen few money or percentage figures to back up user statements. It can be difficult to obtain fair 'before' and 'after' figures. Users may simply recognize that programming has been made easier, and let it go at that.

Here, then, are the benefits of programming work-stations, as described to us by both users and suppliers.

### Increase programmer productivity

Increasing programmer productivity is always the first benefit mentioned, by users and

vendors alike. We have heard it estimated that use of programming work-stations will increase programmer productivity anywhere from 15% to 83%. These increases come from the system taking over some of the programming tasks, thus making programmers more efficient at the tasks they perform.

***Reduce waiting, searching and walking time.*** The people at Four-Phase Systems, who market a stand-alone system that we will discuss next month, estimate that programmers spend 40% of their time waiting—waiting for cards to be punched, waiting for the computer to run their jobs, waiting for the computer to come back up after a crash, and so on. Using stand-alone work-stations, Four-Phase estimates that this waiting time drops to less than 12%.

Programmers seem to know a lot about what other programmers are doing, what types of modules others are writing, etc. So if a programmer thinks he or she might want to use a module that another programmer has spent a lot of time creating, the work-station environment makes this easier to do. The programmer only needs to know the name of the file the module is stored in and if it is publicly accessible. Then he can immediately retrieve that file and review the code, to determine if it can indeed be used. So programming work-stations reduce searching time. They also reduce the time programmers spend searching through their own files, because files are stored electronically rather than in loose paper form.

Having a programming work-station in one's office (or close by) also saves programmers a lot of walking time, particularly to the computer room to submit and pick up jobs. This time savings is not so great if the department has already installed a remote batch terminal. But having a terminal makes keeping track of the progress of jobs much easier.

All of these routine activities have wasted a lot of programmer time. Work-stations reduce this waste.

***Make programmers more efficient.*** The editing, filing, command language, and conversational programming features of work-stations make programmers more efficient at programming.

Coding is speeded up in several ways. Entering procedural code on a keyboard may or may not be faster than having cards keypunched. It depends on the particular programmer. More importantly, the new systems more easily allow programmers to re-use coded modules from other programs.

Likewise, use of pre-coded, pre-tested macro commands and procedures, either supplied by the system or user created, actually cut down on the amount of code that programmers need to write.

Conversational programming features are another method of speeding coding, and eliminating errors at the same time. One system, to be discussed next month, encourages structured programming through conversational programming. For COBOL programming, for example, the system has a syntax guidance feature that displays a menu of admissible instructions for the programmer to choose from at each point in coding.

Even more dramatic increases in efficiency come from non-procedural programming features, which provide most of the code *as well as the program logic.* The programmer concentrates on *what* needs to be done rather than *how* to do it.

Filing features can make locating errors online faster than reading paper listings, particularly on systems that locate the errors for you. And entering corrections at a work-station is much quicker than having new cards keypunched.

So these work-stations increase programmer productivity by eliminating a lot of wasted programmer time and by increasing programmer efficiency during coding and debugging.

### Shorten development time

In addition to making programmers more efficient, programming work-stations shorten software development time by speeding coding and decreasing testing.

***Speed coding.*** We found the non-procedural programming features to be the programming aid that shortens development time the most, because most of the code is generated by the computer. The programmer concentrates on fitting the available pieces together, supplying

the variables, and picking the appropriate options. There are fewer pieces to integrate, so this shortens the design phase also.

Two important questions to ask about a non-procedural programming facility are: (1) Is it general enough and extensive enough to cover most of your programming needs? And (2) Does it produce efficient code, so that programs take up about as much memory space and run about as fast as they would if coded in a conventional manner?

Programming work-stations also speed coding by making it very convenient and easy to access, modify and re-use existing program modules and data definitions. Needless to say, this is much faster than starting from scratch.

Also, some systems have conversational features which partially lead a programmer through coding, by presenting the options available at each point, correcting syntax and punctuation errors, and such. Thus they take over some of the routine features of procedural programming, leaving the programmer free to concentate on logic.

***Decrease testing.*** Use of programming work-stations can also decrease the number of compilation and test runs required, by automatically catching spelling, punctuation, and syntax errors. They also reduce the amount of time programmers need to spend creating job control strings to perform these tests. All of the IBM users we talked with use the on-line files to store standard JCL strings, which they quickly call up and attach to their jobs.

Only one user that we talked with had any figures on how the programming work-station environment had shortened development time. At Buck Knives they estimate that the non-procedural programming features of USER-11 typically cut their development time by a whopping 77%, with an 83% reduction in coding. This reduction is so large, they explained, because they really do very little procedural coding anymore. They mainly co-ordinate USER-11 procedures and specify options.

So programming work-stations shorten development time by speeding coding and decreasing testing.

**Encourage good programming practices**

We have noticed that programming work-stations encourage some good programming practices—ones that may have been impractical or too costly in the past, in the minds of some people.

***More fully documented programs.*** Programming work-stations make it easier to document programs. For one thing, they generate standard program formats automatically. And they make adding comments easier. So one user we talked with has made it a standard practice to comment every line of code, as the code is keyed into the work-station, not afterwards. The editing features make future changes and additional comments easy to add. Another user created a standard macro procedure that keeps the documented version of programs up-to-date with the current production version. Finally, by using existing, documented modules and procedures, the documentation has already been done. We found that systems that have a HELP key or command have well-documented programs.

***Re-usable code.*** We have noted several times that providing easy access to existing program modules encourages re-use of those modules. This is one of the prime benefits that all of the work-station users pointed out to us. They were surprised and very impressed with how much this good programming practice has made life easier for their programmers. And we suspect that these vendor- and user-created modules are also pretty efficient to run. Or at least they become more efficient as other programmers refine them. Programmers and management seem to want to have as many of these standard routines and utility building blocks as possible. Programming work-stations encourage this good practice.

***Throw away code.*** We have often heard programmers and management alike lament that, if only they had enough time to throw away their first version of the code and start over, their resulting systems would be a whole lot better. Well, we found one work-station feature that one company has used to do precisely that. The feature is a command language with flow-of-control commands included in it. These

control commands allow programmers to use the command language as a very high level programming language. Such a command language allows programmers to create programs consisting solely of macro commands. The programs are not efficient to run, but they are quickly written. Once created they can be field tested to see if their designs are correct. Then they can be redesigned, if necessary, and recoded in a more efficient language.

We found powerful command languages on two systems that we will discuss next month—the Programmer's Workbench, developed at Bell Laboratories, and Pet/Maestro, developed by Softlab GmbH in Germany and marketed in the U.S. by Itel Corporation.

Maybe in the future such programming work-station features will encourage more use of throw-away code, a programming practice many seem to want.

***Non-procedural programming.*** It appears to us that non-procedural programming is the next step beyond programming in a higher level language such as COBOL, FORTRAN, BASIC, or Pascal. In non-procedural programming the system leads the user through the task at hand, so it is much easier to write programs. The users do not need to know the intricacies of a programming language. They simply need to know what they want to do and which procedures will help them set up the proper program. The non-procedural feature thus takes programming one step closer to the end user. It makes it feasible for someone in a user department to write some complex, efficient programs with little programming training—*if the system includes the application logic that the user needs.*

Programming work-stations thus encourage some good programming practices that have been formerly difficult or impractical to achieve—more fully documented programs, reusable code, throw away code, and non-procedural programming.

## Enhance program quality

Programming work-stations can enhance program quality by improving system and program designs and by increasing program clarity. Let's look at these.

***Improve design.*** One determinant of program quality is quality of the design. Has the program been structured so that modules perform only one function and so that errors do not ripple from one module to another? Few of the work-stations we saw have much in the way of *system* design tools. Most such first generation programming work-stations are concentrating on coding aids rather than design aids. In the future we expect to see some of the more popular system and program design methods supported—techniques such as SADT, the Jackson Method, Warnier's LCP, and others.

However, one work-station system we saw does support program design by generating structured programming charts, called structograms. These are a type of Nassi-Schneiderman diagram. The system draws the hierarchical diagram, which represents the program's control structure, and the programmer fills in the blanks, which are the program's action items. We expect more graphic aids such as this one to become available in the future.

By easing the coding load on programmers—by making it easier and faster to perform—programming work-stations can, indirectly, improve program design. They give programmers more time to think about design, knowing that coding will move along more quickly.

***Increase program clarity.*** Another determinant of program quality is code clarity. Programming work-stations certainly do enhance clarity. They enforce language-specific coding formats, user-defined procedures, and standard programming conventions, such as line numbering, error detection, file processing, and documentation techniques.

So programming work-stations do improve software quality. And by doing so they reduce program maintenance.

## Reduce program maintenance

All of the above mentioned benefits lead to programming work-stations reducing the amount of program maintenance needed. At Buck Knives they estimate that their maintenance requirements have been reduced by almost 80% using USER-11 procedures. These procedures, which constitute five-sixths of the code in their production systems, need no cor-

rective maintenance. And enhancement maintenance generally involves only re-running a USER-11 procedure and changing the variables. This example, of course, is not yet typical, we believe, but it does show that dramatic reductions in maintenance are possible.

The users we talked with are new users of programming work-stations, because the products are new. They know that benefits such as reduced maintenance are occurring, but they do not know by how much. They see maintenance being reduced because they see fewer errors in programs when existing modules are reused. They see errors being caught earlier. And they see more complete designs. Also the errors that do occur are much easier to find and change. So their use of programming work-stations is reducing maintenance in these ways.

**Improve project management**

Programming work-stations can improve project management in several ways, by increasing visibility of work, by improving team communications, and by providing file security.

***Increase visibility of work.*** Some programming work-stations increase the visibility of project work by providing tracking aids, by maintaining statistics on how many lines of code have been generated or modified in programs, and by keeping logs of jobs submitted to the mainframe. Some keep audit trails of all program changes. And some users have written their own routines, using work-station command languages, to collect pertinent project information.

We also found that the on-line environment encourages coding and testing of single modules. This practice breaks up the development process into smaller increments, which can more easily be tracked by the project manager. Thus, critical situations can be recognized earlier. So work-stations make work more visible to others, to the delight or the disgust of programmers, we do not know which.

***Improve team communications.*** Brook's 'Law' says that adding more people to a late project makes it later, because more inter-communication among project members is required. Communication within a project is very important. And programming work-stations have the potential of improving communications by putting programmers and data processing management into an automated office environment. With the addition of a computer message system, communication among groups can be greatly enhanced. We discussed this subject in the April 1977 and September and October 1978 issues. Computer message systems actually increase informal communication among members of a project, be they down the hall or miles away. Members use electronic messages to record informal messages with others. All members can be sent a duplicate of every pertinent message easily, so all are equally well informed.

Even without the addition of a computer message system, just keeping files on-line and publicly accessible to team members improves communication. We noticed that easy access to such files, along with an extensive command language, encourages programmers to experiment with 'what if' situations. This computerized support augments their discussions with one another and clarifies communication within a team. The command language allows them to quickly formulate all kinds of qustions for the system, and thereby consider more possible alternatives.

So management should look at programming work-stations as an aid to communication as well as an aid to programming.

***Provide file security.*** Most of the work-stations we saw have on-line security features, for accessing files, changing programs, and even running programs. These features allow programmers to define the security levels of their modules as well as of resulting programs. Some levels are: not accessible to any others, read only by others, read and changeable by some others, and program use by specific passwords. The systems that have these types of security features also are very easy to use, generally in a conversational programming fashion.

Having the ability to include security features in programs easily, and having security levels for files are important factors in project management—factors that too often have been neglected because of time considerations. Late projects do not have well thought-out security features. With such features easy to generate,

it greatly decreases a common worry of project leaders.

### Improve working conditions

When programming work-stations replace coding sheets and cards, and perform many routine tasks for programmers, the programmers appreciate the improvement. They like the quicker test response, the syntax checking, the use of abbreviations, and the editing features. And they grow accustomed to the more sophisticated features: conversational programming, command languages, links to data managment systems, and non-procedural programming facilities. These features make programming a lot less tedious.

Some programmers at one company we visited commented that they would change jobs to another company *only if* that company had a programming work-station system as sophisticated as the one they were currently using. Since few such installations now exist, they are not likely to move. All the people we talked with agreed on this point—programming work-stations do improve working conditions, which leads to greater programmer satisfaction and (so far) lower turnover. Programmer turnover is a problem data processing departments have been wrestling with for a long time.

These then are the benefits we uncovered in our study of programming work-stations. Surely these translate into cost reductions over conventional programming techniques. We suggest that data processing management study its programming costs more closely to better evaluate these new systems. We see these systems as providing a more efficient way to develop and maintain software—the wave of the future.

Next month we will discuss several other programming work-stations—stand-alone systems used to develop programs for other target computers. And we will list the major work-station features we found, to help data processing management consider these new systems.

---


REFERENCES

1. *VSE/Interactive Computing and Control Facility, General Information Manual*, IBM (order through your local sales office), GC 33-6066-0, January 1979, price 90 cents.
2. For information about USER-11, contact North County Computer Services, Software Marketing Division, 2335 Meyers Ave., Escondido, California 92025.
3. For information about the H-P 300, contact Hewlett-Packard, 1507 Page Mill Road, Palo Alto, California 94307.
4. We have compiled a list of programming work-station products and systems that we came across in our research. For a free copy of this list, write EDP ANALYZER.



Prepared by:
Barbara C. McNurlin
Associate Editor

## SUBJECTS COVERED BY EDP ANALYZER IN PRIOR YEARS

**1976 (Volume 14)**

*Number*

1. Planning for Multi-national Data Processing
2. Staff Training on the Multi-national Scene
3. Professionalism: Coming or Not?
4. Integrity and Security of Personal Data
5. APL and Decision Support Systems
6. Distributed Data Systems
7. Network Structures for Distributed Systems
8. Bringing Women into Computing Management
9. Project Management Systems
10. Distributed Systems and the End User
11. Recovery in Data Base Systems
12. Toward the Better Management of Data

**1977 (Volume 15)**

*Number*

1. The Arrival of Common Systems
2. Word Processing: Part 1
3. Word Processing: Part 2
4. Computer Message Systems
5. Computer Services for Small Sites
6. The Importance of EDP Audit and Control
7. Getting the Requirements Right
8. Managing Staff Retention and Turnover
9. Making Use of Remote Computing Services
10. The Impact of Corporate EFT
11. Using Some New Programming Techniques
12. Progress in Project Management

**1978 (Volume 16)**

*Number*

1. Installing a Data Dictionary
2. Progress in Software Engineering: Part 1
3. Progress in Software Engineering: Part 2
4. The Debate on Trans-border Data Flows
5. Planning for DBMS Conversions
6. "Personal" Computers in Business
7. Planning to Use Public Packet Networks
8. The Challenges of Distributed Systems
9. The Automated Office: Part 1
10. The Automated Office: Part 2
11. Get Ready for Major Changes
12. Data Encryption: Is It for You?

**1979 (Volume 17)**

*Number*

1. The Analysis of User Needs
2. The Production of Better Software
3. Program Design Techniques
4. How to Prepare for the Coming Changes
5. Computer Support for Managers
6. What Information Do Managers Need?
7. The Security of Managers' Information
8. Tools for Building an EIS
9. How to Use Advanced Technology
10. Programming Work-Station Tools

*(List of subjects prior to 1976 sent upon request)*

## PRICE SCHEDULE (all prices in U.S. dollars)

| | U.S., Canada, Mexico (surface delivery) | Other countries (via air mail) |
|---|---|---|
| Subscriptions (see notes 1,2,4,5) | | |
| 1 year | $48 | $60 |
| 2 years | 88 | 112 |
| 3 years | 120 | 156 |
| Back issues (see notes 1,2,3) | | |
| First copy | $6 | $7 |
| Additional copies | 5 | 6 |
| Binders, each (see notes 2,5,6) | $6.25 | $9.75 |
| (in California | 6.63, including tax) | |

NOTES

1. Reduced prices are in effect for multiple copy subscriptions and for larger quantities of a back issue. Write for details.
2. Subscription agency orders are limited to single copy subscriptions for one-, two-, and three-years only.
3. Because of the continuing demand for back issues, all previous reports are available. All back issues, at above prices, are sent air mail.
4. Optional air mail delivery is available for Canada and Mexico.
5. We strongly recommend AIR MAIL delivery to "other countries" of the world, and have included the added cost in these prices.
6. The attractive binders, for holding 12 issues of EDP ANALYZER, require no punching or special equipment.

Send your order and check to:
EDP ANALYZER
Subscription Office
925 Anza Avenue
Vista, California 92083
Phone: (714) 724-3233

Send editorial correspondence to:
EDP ANALYZER
Editorial Office
925 Anza Avenue
Vista, California 92083
Phone: (714) 724-5900

Name_____

Company _____

Address _____

City, State, ZIP Code_____