EDP ANALYZER

© 1979 by Canning Publications, Inc.

NOVEMBER, 1979
VOL. 17, NO. 11

STAND-ALONE PROGRAMMING WORK-STATIONS

Last month we discussed one type of programming work-station—host programming work-stations. These are software and firmware tools for use on in-house host computers. We described how users are obtaining increased programmer productivity and improved quality of programs through the use of these tools. This month, we look at a second type—stand-alone programming work-station systems. And we develop a checklist of system features which data processing management can use to better evaluate these up-and-coming products.

Pullman Kellogg is an international process engineering company, specializing in building oil refineries, fertilizer plants, and other large process control factories. It is a subsidiary of Pullman Inc. and employs some 3500 people worldwide.

Several years ago Pullman Kellogg centralized its data processing operation at its headquarters in Houston, Texas. There it uses the computer equipment of another Pullman subsidiary, Pullman Computer Services. Pullman Computer Services operates two computer centers in Houston, each with two IBM 370/158s that use TSO and CICS. Kellogg has its own system development staff of 85 programmers.

After moving to Houston, the Kellogg programmers were encouraged to use TSO for on-line programming. Management speculated that this would increase their programmers' productivity by about 15% over their previous card and batch development environment.

Well, the switch to TSO was successful—so successful in fact that, by early 1978, it had degraded system response time badly. And their 15% increase in productivity was evaporating. Programmers were waiting anywhere from three to twenty minutes to just sign onto the system. Also, TSO usage charges had grown rapidly.

These factors prompted the lead technical support person to present the problem at a monthly 'open forum' meeting. These are meetings at which members of the department present and discuss current departmental problems with the data processing manager. The discussion led to formation of a six-member team to investigate alternatives to TSO for on-line software development and maintenance.

After several months of intermittant study, the team recommended acquiring a Four-Phase Programmer Workstation (PWS). PWS is a stand-alone programming work-station system. The prospect of off-loading all of their development work, except compilation and testing, from the large IBM machines appealed to Kellogg very much, particularly because of the large TSO costs they were encountering.

ISSN 0012-7523. Multiple copy prices listed on last page. Photocopying this report for personal use is permitted, providing payment of $2.00 fee *per copy of report* is made to Copyright Clearance Center, P.O. Box 765, Schnectady, N.Y. 12301; please include identifying fee code 0012-7523/79/110001-13$02.00.

Kellogg was familiar with Four-Phase because they had been using its data entry system for some time, and were pleased with it. So in September 1978 a twelve-station PWS system was installed. The system currently includes a Four-Phase IV/90 mini-computer with 192k bytes of memory, two 67.5 megabyte disks, 600 and 1000 line per minute printers, a card reader and 15 display work-stations.

Due to the large number of programmers at Kellogg, they decided to perform training in-house, and as quickly as possible. So the training director held three two-hour sessions a day. With six programmers in each session, the entire department was trained in two weeks late last year. No further formal training has been needed.

Migration from TSO to PWS occurred within two weeks, because the programmers preferred the rapid response time on the off-line system. And management was pleased with the cost of PWS. It has a fixed cost of about $1300 a week for twelve work-stations; the cost is not usage dependent like that of TSO. TSO usage dropped from 2,500 CPU minutes per week to about 5 CPU minutes. Management also believes that they have regained their projected 15% increase in programmer productivity, and perhaps more.

For software maintenance, PWS is used as follows. The programmer walks to one of the work-stations, which are located at different points in the department. After signing on, he or she uses a PWS macro command to create a copy of the source listing of the program he or she is to work on. The source code library is maintained on the mainframes by Panvalet, a program librarian package. Since this copying function may take anywhere from one minute to 15 minutes, the programmer signs off or performs other work on the work-station.

When the program listing is available on PWS, the programmer makes the needed changes on-line using the system's editing features. After these are completed, he executes a canned job control routine to compile and test the program on the target host, the IBM 370/158. This PWS routine performs all of the necessary operations to place the job in the host's RJE batch queue, return the run results to PWS, and place the program in Panvalet's test library. The programmer can then view the results of the test run on the work-station.

For creating new programs on PWS, the programmers often first retrieve strings of JCL instructions, CICS maps for database use, or COBOL modules. Using these, they delete what is not needed, add new information, and thus more quickly create new programs. This procedure actually leads the programmers through the coding phase, so they tend to make fewer omission errors, we were told.

Pullman Kellogg has not found it necessary to install work-stations in every programmer's office. They have found one work-station for every six programmers is sufficient. The programmers readily share the work-stations and there is little waiting to use one. The people at Kellogg say this is because use of PWS makes coding and editing sessions shorter and more efficient than had been true using TSO.

Pullman Kellogg is very pleased with their use of PWS. And their enthusiasm for the on-line approach is spreading to other Pullman divisions.

BNR INC

BNR INC is a subsidiary of Bell-Northern Research Ltd., which has its headquarters in Ottawa, Canada, and is a research and development laboratory for Northern Telecom and Bell Canada. Bell-Northern Research is one of the largest telecommunication research laboratories in the world, employing some 2800 people. BNR is located in Palo Alto, California, just south of San Francisco, and employs about 300 people. BNR does telecommunication and automated office research. For example, they develop the software for computerized PBXs manufactured by other Northern Telecom subsidiaries. BNR has three DEC PDP 11/70s, one 11/60 and one DEC 2050 on site. They also communicate to Bell-Northern Research's IBM 3033 in Ottawa over a leased line.

In 1978 BNR wanted to increase computer support for its various programming projects. Since they were operating their equipment under the UNIX time-sharing operating system developed by Bell Labs, and since they liked it very much, they became interested in a similar Bell-developed operating system, PWB/UNIX.

This system is commonly called the Programmer's Workbench.

The Programmer's Workbench is based on UNIX. It contains most of the UNIX features and has enhancements designed specifically to support software development. Western Electric (part of the U.S. Bell family system, with no connection to Bell Canada or Northern Telecom) licenses use of both UNIX and PWB/UNIX, but they do not support either system. So they allow their licensees to enhance, market and support both of these products. One such licensee is Interactive Systems Corporation (ISC) in Santa Monica, California, near Los Angeles.

In mid-1978 BNR decided to license use of Interactive System Corporation's version of the Workbench (the IS/1 Workbench), because it could run on both their PDP 11/70s and their 11/60. Further, they wanted ISC's training and support for the system.

BNR also installed some complementary products developed by ISC. One is INed, an enhanced editing package that provides split screening, cut-and-paste operations, and full-screen editing.

Another ISC product that BNR uses is the INtext terminal. When used with the INed package, most of the editing operations are carried out in the terminal processor, thus reducing the workload on the DEC mainframes. BNR has about 36 of these terminals and about 65 other terminals to support 200 users. Some 125 of these users are programmers; the other 75 are office personnel who use only the word processing features of the systems. BNR has found the INtext terminals to be very cost effective. They estimate that these have taken over about one-half of the work of their mainframes.

A third complementary product is the remote job entry (RJE) sub-system. It handles the transmission of jobs to target systems. BNR uses the RJE facility to transmit jobs to the IBM 3033 in Ottawa.

Use of the Workbench for software development at BNR generally begins in the requirements phase. BNR puts all requirements documents on the system, using another product, INroff. INroff is a text formatting system for use with line printers, matrix printers, and typewriter-like devices as well as typesetters. It allows users to define standard formats to be used for different types of documents. These formats can then be used by calling their macro command names.

BNR's requirements documents have a standard format, making them easier to review by the users—the engineers. And they are kept up-to-date with the system's text editing features. On one project, the requirements document went through 15 iterations in just one day. The secretary was able to keep up with this pace using the system.

BNR finds the Workbench to be a very sophisticated programming tool. They took ISC's four-hour training course, but some users found the system hard to use at first. Now, after some months of use, they greatly appreciate its sophisticated features.

One such feature is the Workbench's command language, known as the 'shell.' The shell is used for entering user commands to the UNIX operating system. In the Workbench, the language's capabilities have been substantially extended, making it convenient to use as a very high level programming language. The commands can be used alone, strung together on one line to form a very powerful sequence of operations, or combined with control commands to form shell programs.

BNR programmers do a lot of shell programming, we were told. They create standard RJE JCL strings for the 3033 and place them in files. T. A. Dolotta, *et al* (Reference 1) point out that programmers who use the shell for this purpose make 20% fewer errors than those who write their own RJE JCL.

These authors also point out that shell programming is often preferred by programmers, because the commands are quickly learned. Anyone who has used the UNIX system has learned some of these commands. And shell programming is fast, requiring much less human effort than conventional programming languages do. If a program is going to be used a lot, it can be recoded in a more efficient language, after the shell program has proven its worth. For programs not run often, the original shell programs suffice.

In at least one case, a BNR programmer has used the shell to write throw-away code for a significant application system—an in-house message system. The programmer wrote the programs using the shell commands. It was not a very efficient system to run, but it was created quickly. Then the system was used for about two months by some half dozen users. Their critiques were evaluated and the system was redesigned where needed. Then, it was entirely recoded in the language C, the UNIX system implementation language. And the old shell system was thrown away. The final system consists of 5000 lines of code, yet the entire project took only about four work-months of effort to complete. More importantly, the usefulness of the new system was demonstrated one-fourth of the way through the project.

Another feature of the Workbench that BNR uses a lot is the source code control system (SCCS). SCCS is a set of commands that is used to control changes to source code and files of text, such as manuals. Using it, versions of source code are tracked, along with the date of each change, who made each change, and why. The system stores the common code once, and the various versions separately, so that any version of a program can be recreated, as desired. BNR uses SCCS to track and control most of their software projects, so they have a complete history of each project's work.

From their use of the Workbench, the people at BNR have found that it has increased their productivity as well as the quality of their software. They think that because of its sophisticated features, the Workbench is aimed at people who use it constantly, such as programmers.

Two users of Pet/Maestro

One of the leading programmer work-station systems has been developed by Softlab GmbH of Munich, Germany. In Europe, this system is marketed by Softlab under the name PET/X1150, and it runs on the Philips X1150 minicomputer equipment. In the U.S., it is marketed by Maestro Systems, Inc., a subsidiary of Itel Corporation, under the name 'Maestro.' We will refer to the system as 'Pet/Maestro.'

Since Pet/Maestro is among the most powerful of the programmer work-station systems and since the longest usage of this system has been in Europe, we visited two organizations there to learn of their experiences.

Vereins-und Westbank

Vereins-und Westbank, with headquarters in Hamburg, is the largest regional bank is northwest Germany. The bank has 265 branches, located mainly in the northern part of the country. At the end of 1978, deposits were almost DM 8 billion.

Hamburger Dataverarbeitung GmbH (HDV) is the wholly-owned data processing subsidiary of Vereins-und Westbank. HDV provides data processing services not only for the bank and its branches but also for a number of bank customers. HDV uses an IBM 370/158AP, with 24 Philips remote job entry terminals. There is a development staff of 65 people, the majority of whom are programmers.

HDV has been a rapidly growing operation, and additional workspace has been needed almost continually. So, for some years, the programmers have not been located near the computer center, usually being 6 to 10 km away. Initially, source decks had to be transported physically, but for several years a remote job entry terminal had been used for communicating with the host computer. But even with the RJE terminal, there still were problems. Turnaround was slow during peak periods, for instance, or the host might be down, etc.

In early 1977, several of the managers at HDV saw an announcement of the new Pet/Maestro system. They had been looking for an improved support tool for programming and had looked at IBM's time-sharing option (TSO). But the operating system they were then using did not support TSO so that was not a solution they could use. They checked on the new Pet/Maestro system, liked what they saw, and got HDV to enter an order for two systems. Each system was to have eight work-stations and 2.5 million bytes of disk storage. These systems were installed in May 1977.

The acceptance of these new systems was almost immediate, we were told. All of the prospective users were given a four-day training course. Within two months, both systems were fully loaded and most of the programmers were asking for their own terminals. (A few of the programmers initially felt that the systems were downgrading them to 'key punchers,' but after some use, they changed their views.)

So in October, 1978, HDV ordered three more Pet/Maestro systems, with a total of 50 work-stations, and increased disk capacity to 62 Mb per system. And in August of this year, they increased the total number of work-stations to 68.

Most of HDV's existing applications are batch systems that have been written in assembler language. But new systems are being written in COBOL, and a growing number of them are interactive applications that use IMS.

Use of Pet/Maestro. HDV uses Pet/Maestro much like a typical on-line system—with some important differences based on features that Pet/Maestro offers. A text editor is provided, with which programmers enter and change lines of code. A library facility is included—with access control—for storing code in an organized fashion, for easy retrieval. A programmer can copy sections of already-written code, or standard data definitions, into a new program. A reconstruct facility is provided, along with a number of other programming support features. In short, HDV uses an integrated set of functions, provided by Pet/Maestro, in the program development process.

In addition to the various functions provided with the Pet/Maestro system, HDV has developed about 55 procedures of their own, using the command language, for program development functions they desire.

When a module has been coded, the programmer instructs the Pet/Maestro computer to transmit the module to the host 370 system, on a remote batch basis, for compiling and test. The results are transmitted back to the Pet/Maestro system, often within 30 minutes—but turnaround may be in the order of two hours during afternoon peak load periods. The programmer can examine the results via the work-station, make any corrections using the text editor, and repeat the compile and test, if required.

Other uses. This use of Pet/Maestro by programmers is only part of HDV's use of the systems. As it was expressed to us, "We are using these systems almost as 'automated office' systems." Here is a very brief summary of those uses.

The systems are used by analysts and designers for storing requirements, specifications, and design statements, in text form. The text editor makes it easy to keep these statements up to date.

Users can send messages from any work-station to any other, via the host.

The systems are used for recording and keeping current certain management information, such as the organization charts, staff lists, job assignment lists, and so on.

And the results of a business systems planning (BSP) study made last year are kept in the systems.

Some benefits. We asked about the benefits that have been obtained. Programmer productivity has been improved by at least 20 to 30%, they estimate, both for maintaining and enhancing existing systems as well as for developing new ones.

Pet/Maestro has removed some workload from an already loaded host CPU. And by eliminating most of the syntax errors, it has cut down the number of compilations.

The programming effort is now much less dependent on the host computer; it does not stop if the host is down. On the other hand, the programming effort is *very* dependent on the Pet/Maestro systems; if one of those is down, the people using it essentially stop work.

But the overall impression we received is that HDV feels it is just beginning to exploit the capabilities of the Pet/Maestro system. Both as the system is enhanced by Softlab and as HDV extends its usage, benefits will continue to accrue.

Enka BV

Enka is the largest division of Akzo NV, an international group of industrial companies that employs over 83,000 people world-wide. The group's headquarters is in Arnhem, The Netherlands. The Akzo companies produce man-made fibers, salt, heavy and specialty chemicals, and other products.

The Enka division produces most of Akzo's man-made fibers, in the form of textile yarns and fibers, industrial yarns, and miscellaneous polymer products. The division has two main components—Enka AG in Wuppertal, West Germany (which is the division headquarters) and Enka BV in Arnhem, The Netherlands. We visited Enka BV to learn about their use of Pet/Maestro.

Until early 1978, Enka used RJE terminals to communicate between the programmers and the Akzo IBM 370 host (now a 3033). Turnaround time for compilations and tests averaged between one and two hours. But there were the usual problems with this type of operation and the people at Enka were looking for better methods.

So they began an investigation of the use of IBM's TSO for on-line program development. A pilot project, involving four programmers and two terminals, was set up. While it was an improvement, TSO still was not what they were seeking, they found; use of it required a good knowledge of the system, it was rather expensive, and response time varied from three seconds (acceptable, unless you are scrolling through files) to five minutes (completely frustrating), depending upon the load on the host computer.

Then in late 1976, one of the Enka software specialists saw a demonstration of the Pet/Maestro system in Dusseldorf, Germany. He was impressed, but found that the system was not yet available in The Netherlands. In fact, it took Enka over one year to get their first Pet/Maestro system. The first one was installed in Arnhem in February 1978, while the second was installed in Wuppertal shortly thereafter.

The original system at Arnhem had 10 work-stations for a user group of 35 people. It was brought in for a trial period of three months. Softlab gave a three-day training course to the staff two weeks after installation, but by that time some of the staff had already learned to use it on their own.

What was the response to the three-month trial? In the words of the software specialist we talked with, "We found that we couldn't take the system out. Just about everyone was enthusiastic. I made a survey, and one of the responses was amusing. This programmer said, 'If anyone tries to take my terminal away, I'll kill him.' Most of the other responses were just as positive but not that aggressive."

In December 1978, Enka BV installed its second Pet/Maestro system and now has a total of 16 work-stations. Enka AG has one system with 10 work-stations.

Use of Pet/Maestro. At Enka BV, system analysts use Pet/Maestro for developing the specifications for a new system in text form. The top-down design features, plus the text editor, make it easier for them to correct and add to the specifications. Working from the specifications, the programmers then use the system's interactive structured program design feature for developing program modules.

With the structured program design feature, a programmer first indicates that he (or she) wishes to create a module. This starts a dialog between the system and the programmer. For each step, the system gives the programmer five choices: an action, a loop, the end of a loop, a branch point, or a case construction. As an example, the top level definition of a module might consist of the following series of statements: (1) Action—open personnel master file; (2) Action—validate file expiration date; (3) While—not at end of personnel file; (4) Case—(a) applicant resume received, (b) interview, (c) hire; (5) End while. The programmer would then go on to give more details for each of the three case situations.

At this point, the programmer asks Pet/Maestro to draw a 'structogram' of the logic. The structogram is a form of Nassi-Schneiderman diagram. It is usually printed out, so that the programmer can study it and go over it with the system analyst and, if necessary, the user. If changes are needed, they are made in the statements and the system draws new diagrams.

With the module logic developed, the programmer then proceeds to code the module in COBOL and transmit it to the host for compilation and test. To aid in program development, Enka has created about 40 of their own macro procedures, in addition to the standard functions and procedures included in Pet/Maestro.

Some benefits. It is hard to compare the new environment with the old one, we were told, because it is difficult to get comparable figures for the old situation. But the software specialist we talked with had made a survey among the whole development staff. Following are the average estimates found from this survey. Overall programmer productivity has increased about 30%. For program maintenance, productivity gains are even higher, in the order of 40%, due to the ease with which changes can be made. Quality of the work done has increased, and users have estimated this increase to be 10% to 15%. The documentation is more complete and more up-to-date. And the system gives fast response. Independent of the number of users on it, the system gives a response time of between one-half and four seconds, depending on what is asked of it. Users of Pet/Maestro are under no pressure, since "the meter is not ticking" and the user can work at his/her own pace. Further, the (fast) system response time does not slow down the user.

Programming work-station types

To repeat ourselves a bit from last month, we define programming work-stations as those portions of computer systems that have been developed specifically for use by programmers for software development and maintenance. To use them, programmers work at work-stations, either CRT displays or typewriting terminals, rather than with pencils, coding sheets, cards, and paper listings.

We see two main types of programming work-stations: host programming work-stations and stand-alone programming systems.

Host programming work-stations, which we discussed last month, are software or firmware products designed to run on a company's main computer. We described three such systems last month. One is aimed mainly at enhancing procedural programming, when conventional programming languages are used; it also supports one non-procedural language. Another increases productivity through non-procedural programming, where the programmer specifies *what* needs to be done, rather than *how* it is done. The third system is a firmware system for a departmental-size mini-computer, to make it easier for the user department to develop programs.

Stand-alone programming systems, on the other hand, are devoted entirely to program development support. Programs developed on these systems are generally intended to run on different machines, called 'target systems.'

For a list of the products we found in our research, see Reference 7.

Let us now look at stand-alone programming systems in more detail.

Stand-alone programming systems

Stand-alone programming work-station systems are relatively new offerings. They are complete, stand-alone systems designed specifically to support software development. As such, the programs written using them are generally intended to run on target machines. So these stand-alone systems require a communication facility for sending programs to the target computer for compiling and testing.

Karl Drexhage, a management consultant who has had extensive experience with programming work-station use in Europe, points out in a paper (Reference 6) some of the benefits of stand-alone systems over host-resident systems. For one thing, he says, like distributed data processing, they shift work from a central computer onto a distributed computer—in this case, a dedicated development computer. And since development work no longer competes with production work for system resources, stand-alone systems give better response time. They give a low guaranteed response time—in the order of two seconds—while host system response times vary with the workload (and can become intolerably slow during peak usage).

Being off line, data security is better, he says. And these systems are more reliable since there are fewer parts to break down. They are designed for development work, so they are better human engineered. And stand-alone systems are lower in cost, because they eliminate the need for added memory, disk storage, communication software, and terminals required for host-based systems. Finally, says Drexhage, they are host independent, so they can be used to develop applications for different hosts.

Let us now look at the three stand-alone systems mentioned earlier, in a little more detail: PWS from Four-Phase Systems, Programmer's Workbench developed at Bell Laboratories, and Pet/Maestro from Softlab GmbH in Europe and Itel Corporation in the United States.

PWS from Four-Phase Systems

The Programmer Workstation (PWS) from Four-Phase Systems is a stand-alone system designed specifically to support software development for IBM System 360, 370, and 3000 Series computers. Users can develop programs in any language that these target mainframes can compile.

The typical basic system consists of a Four-Phase mini-computer with 96k bytes of memory, a 67.5 megabyte disk drive, a 300 line per minute printer, a communication controller, and up to 16 display work-stations. It can be upgraded to 192k bytes of main memory, with other peripherals also added.

To compile or test a program, the programmer places the job and its associated JCL string into a PWS SEND queue. The system transmits the files over communication lines to a batch queue in the target mainframe. Following execution, the mainframe sends the results back to the PWS printer. This output can then be retrieved at the work-station.

PWS allows editing in both command and full-screen modes. In the command mode, the cursor is outside the display window that contains the text. From here the user can type in commands to work on lines of text or entire files. The user can do such things as: (1) request automatic tab settings and format control for writing Assembler, COBOL, FORTRAN, and PL/1 programs; (2) search for strings of text, change and delete portions of the text, and then get successive occurrences of that string; (3) scroll forward one or ten records at a time; (4) perform functions on entire files, such as delete, rename, print, save, and queue for transmission; and (5) monitor the status of jobs sent to the target machine.

In full-screen editing mode, the cursor moves into the 21-line window containing the text. The user can directly edit the data using function keys. PWS also provides programming function menus of often-used functions.

(Note that Pet/Maestro, which also runs on Four-Phase equipment, is a different system.)

For more information on the Four-Phase PWS, see Reference 2.

Programmer's Workbench from Bell Labs

As we have mentioned, the Programmer's Workbench (technically named PWB/UNIX) is a time-sharing operating system developed at Bell Laboratories. It runs on DEC PDP-11 computers, models 45 to 70. (Some licensees have parts of it running on smaller DEC machines, we are told.) Also, Interactive Systems Corporation (Reference 4) offers VAX/WB, to run on the DEC 32-bit VAX machine under the VMS operating system.

Programs developed using the Programmer's Workbench can be targetted to run on DEC machines using UNIX, on IBM System/370, and on Univac 1100-series computers. Some licensees also include Burroughs in this list. The Workbench can act as either a stand-alone programming system or as a host system (for testing and compiling programs as well as running production jobs).

The Workbench contains a hierarchical file system of directories and files. Use of the files is very flexible and contributes to the Workbench's usefulness. There are nine file protection modes possible—all combinations of read, write, and execute access, for three types of users: the file creator, a specific group of users, such as his project team, and all other users. The protection status of a file can be redefined by the creator at any time.

We have already discussed the system command language, the shell. Any program written using it can be named and used as if it also were a command. The output of one command can be connected to the input of another, so complex operations can be created by chaining together operations of simple programs.

For example, there are shell commands which can: (1) search a file, or several files, for a given string of characters; (2) compare two files and list the differences; (3) check a file for mis-spellings; (4) format a file based on instructions embedded in the test; and others. With a single command, a user can format a body of text, add line numbers, double space the text, and direct the result to an output device.

The Workbench also includes powerful word processing functions, including text formatting commands, spelling error detection, and numerous text editing functions, such as automatic pagination, hyphenation, right margin justification, footnote placement, a multi-column page option, a table of contents generator, and different paragraph styles.

The RJE facility that we mentioned earlier makes the Workbench look like a card reader/punch and line printer to the target machine. The RJE sub-system performs all of the functions necessary for communicating with the target, such as gathering together the necessary files, transmitting the job, receiving the completed work, opening a new Workbench file into which this output is put, etc.

Another feature of the Workbench is its two test drivers. These allow the system to simulate interactive terminals operating either on a Univac 1100-series computer or an IBM System/370. These are especially useful for complex testing situations, such as interactive database management and data communication operations.

For more information on the Programmer's Workbench see References 3 and 4.

The Pet/Maestro system

In our discussion above of HDV and Enka, we have given many of the characteristics of the Pet/Maestro system. So we will simply summarize here some of the key features of the system. For more information, see Reference 5.

Pet/Maestro has been designed to run on a mini-computer, independent of the host computer for which programs are to be written. In Europe, Pet/Maestro is marketed by its developers, Softlab GmbH, and runs on Philips X1150 equipment (actually a Four-Phase mini-computer). In the U.S., it is marketed by Maestro Systems Inc., a division of Itel Corporation. It runs on Four-Phase equipment. (This is a completely different work-station system from the Four-Phase PWS, described above.) Pet/Maestro works with any host computer that supports a normal remote job entry protocol, we are told.

Each Pet/Maestro system can currently handle up to 20 work-stations. Each system is free-standing. Within one system, information can be transferred between work-stations. Communications between work-stations on different Pet/Maestro systems are handled via the host computer.

Pet/Maestro supports program development starting with specifications and continuing through module design, coding, and testing. It allows programming in most of the popular languages, including COBOL, PL/1, FORTRAN, ALGOL, and others. Documentation can be both created and maintained throughout the process.

To illustrate the use of Pet/Maestro, consider the case of the programmer writing a COBOL module. The programmer indicates that he/she wants to create a COBOL program, whereupon Pet/Maestro gives the list of the four COBOL divisions and asks which one is desired. Assuming that the procedure division is requested, and skipping some of the details, the system asks for the first verb (for the first statement). When it has been entered, the system gives a list of the options from which the programmer must choose, for that type of statement. The system then generates the fixed portion of the statement for the selected option, and prompts the programmer for the variable information (data names, etc.). Because of this approach, very few syntax errors occur, we were told by users. When the coding has been completed, the programmer instructs the Pet/Maestro computer to transmit the module to the host computer, on a remote batch basis.

A text editor is provided for making additions, deletions, changes, and global replacements in a body of code. But the text editor is also useful for handling textual information, such as narrative requirements statements, system and program specifications, and so on.

A Pet/Maestro system provides a variety of functions to support the programming process. It encourages structured programming by providing five control constructs for designing a program (action, loop, end of loop, branch, and case). It draws Nassi-Schneiderman diagrams of the logic that has been expressed in terms of these control constructs. It allows for the easy marking of sections of code, text, and data—and these sections can be indexed. Moving forward and backward, or from section to section, is done by one key stroke. A shorthand capability allows programmers to define their own abbreviations for often-repeated words, data names, and such; the abbreviations can be recalled by depressing a function key. The system allows users to easily copy selected portions of existing code into new programs that they are working on. An audit trail is kept of all changes to each program, which is becoming a necessity in some countries where new privacy laws require the ability to reconstruct a previous version of a program.

In addition to these and other standard features, Pet/Maestro provides a procedure (or command) language whereby users can create their own functions. This language is easy to learn, we were told, but is a bit different from conventional languages; it uses pre-defined variable names, for instance. At Enka BV, a procedure has been created for drawing flowcharts, for system and program design purposes. We saw a user create a 10-box flowchart in less than one minute, as an illustration of the ease of use.

Pet/Maestro provides project supervision support. It maintains statistics on how many lines of code have been generated and/or modified by each system user, for instance.

A decision table feature has been announced but had not yet been received by the users we talked with. With this feature, a programmer would build a decision table that gives program logic, in a dialog fashion. The system checks the table for incompleteness, redundancy, and inconsistency.

Pet/Maestro has recently provided the '3270 mode' function. By using just one key, the Pet/Maestro terminal can be converted to 3270 mode, for operating with TSO—say, for on-line debugging on the host. The terminal can be converted back to Pet/Maestro just as easily.

Checklist of features

Judging from our brief product descriptions both last month and this month, it is quite obvious that no one system has every desirable feature. Such a system undoubtedly would be very expensive. And maybe having lots of 'bells and whistles' is not what every data processing department needs.

To help data processing management better evaluate programming work-stations, we have compiled a fairly complete list of system features, compiled from existing systems. We suggest that management and programmers classify these features in their order of importance and then begin searching for a system that has the desired features.

These then are the major features we found in programming work-stations, *listed in alphabetical order.*

Audit trails. Some programming work-stations have facilities for keeping track of software changes. For those that do not, a source code library package can often be acquired to perform this function. These facilities have different features, and it is really up to each department to decide how elaborate a log of changes they want to keep. As mentioned, some new privacy laws require that a previous version of a program be reconstructable; audit trails make this requirement much easier to live with.

Command languages. The most basic type of command (sometimes called 'procedure') language that we saw has 20 to 30 macro commands supplied by the vendor. These are generally basic utility programs that can be called upon to run by using their names. Typical ones are sort, send output to printer, send job to mainframe, copy file, etc.

Systems that provide these macro commands usually also allow users to write routines, give each routine a name, store it in a file, and invoke it by calling its name. Another capability that may or may not be provided is the ability to string these commands together in a sequence to form a macro procedure.

On a more sophisticated level (and some users say a more useful level) are command languages with a hundred or more macro commands. They usually also contain control commands, thus becoming very high level programming languages. The commands generally are interpreted, not compiled, so they are not as efficient to run in a production environment. But they give very fast response time in a development environment.

Communication links. During programming, project members will need to communicate with the host, with other team members, and possibly with other system facilities. For communicating with the target computer, the stand-alone work-station systems should provide the necessary links. They generally provide two options, communication over telephone lines or generating magnetic tapes. For communicating over telephone lines, these work-stations have a front-end communications controller, making them appear as remote batch terminals. Speed of transmission is important because the communication task can tie up the work-station system and degrade terminal response. The RJE facility is necessary not only for compiling and testing programs but also for retrieving and sending programs to the source code library on the target.

Message systems are a feature we think users will find most useful for keeping all team members equally informed. One system has a communication file capability built in. One line of the work-station display screen is reserved for use of this feature. Another system has a complementary message system package that can be added to it.

Some systems have links to other features, such as typesetters, the corporate database management system, etc.

Debugging aids. We saw a wide range of system capabilities aimed at reducing the amount of time programmers spend debugging programs.

For syntax corrections, some systems spot syntax errors and alert the programmer while code is being entered. Following compilation, one system automatically reports the probable location of the errors. And these features appear easy to use.

Several systems allow interactive debugging. That is, the programmer is alerted to an error during compilation or test execution, and he or she can correct the error and continue the test. Other systems do not allow this. Once a job is submitted, it is executed in the background mode and the programmer uses the work-station for other work. One person we talked with had doubts about the value of interactive debugging, saying that its overhead cost does not justify its usefulness for most programmers.

For testing purposes all of the systems allow creating test data files which can be called up for testing. Seeing a demonstration of test data creation can help prospective users see how much the editing and formatting features of each work-station help in this job.

One system contains test drivers for simulating some interactive terminal situations. One driver simulates a Teletype cluster controller with up to four terminals, for testing programs on Univac 1100-series computers. The other driver simulates one or more IBM 3270 cluster controllers, each controlling up to 32 terminals.

Conversational programmming aids. In conversational programming, the system may help the programmer create conventional programs, by automatically generating formats, presenting coding options, numbering lines, etc. We found conversational aids available for designing modules, writing program and control code, and for generating decision tables and structured flow charts.

Design aids. All of the systems we saw have concentrated on providing programming aids. Most have not yet implemented graphical design aids for system design. One system does have a graphic aid for program design. The system generates a type of structured flowchart, based on a set of high level pseudo-code statements entered by the programmer.

This system also prompts the users for creating decision tables. And the system verifies that the table is complete, with no inconsistencies or redundancies.

Documentation aids. All of the systems try to make it as easy as possible for programmers to include comment statements while coding, and to add and delete such statements afterward. Most systems provide formatting features, tailored to specific languages. Some systems have a HELP function, to aid programmers and end users when they do not understand a system request. These are like on-line reference manuals.

Editing capabilities. The editing capabilities of the work-stations are the most obvious and often the most touted features. We found that the capabilities varied widely.

The most easily used editing functions are those associated with function keys—that is, press the key and the function is performed. One system goes so far as to allow every keyboard key to have a function, when pushed in conjunction with a function control key. Another has eight function keys along side the display screen, with the functions changing as needed. Most of the systems have special keys for common functions: cursor control, delete character, insert entry, etc. Editing functions not performed with keys are performed by typing commands, such as global replace, search, copy file, change, etc. The systems varied in the number of such commands available.

Additionally, some systems have some automatic editing features, such as punctuation, spelling and syntax error correction.

Expandability. Two points should be mentioned. One, there is a good chance that every programmer will eventually want his/her own terminal. This should be kept in mind when selecting a system. Secondly, for host systems, as terminals are added, the host will load up—and response times will suffer. Eventually, one or more stand-alone systems may be desired.

Extensibility. The user should be able to add features not contemplated by the work-station designers, but essential for the user's use. Check to see if you can add such features.

File facilities. We found a great diversity in the file facilities of the various programming work-stations. As a baseline, all of the systems provide each programmer with: (1) a file area for keeping program modules, test data, job control instructions, etc., each with a name for easy retrieval; (2) a directory of accessible user files and publicly available files; (3) a work area for constructing new programs, editing old programs, debugging, etc.; (4) file protection features; and (5) file macro commands for manipulating the library contents, such as copy file, rename it, delete it, and so on. Some systems offer several options in each of these areas.

Source code maintenance features vary in the systems also, from simple updating to keeping logs of program versions, specifying changes made, who made changes, etc.

One last point about files and storage. Dolotta, *et al* (Reference 1) note that all time sharing systems are continually short of disk space. And the RJE facility, which dumps a lot of information into a programming work-station rapidly, accentuates this problem. We noticed that the users we visited opted for larger disk storage than the basic systems provided.

Non-procedural programming features. In non-procedural programming the programmer concentrates on deciding *what* needs to be accomplished, and the system generates the 'how to do it' code. Although this is not a new concept, it seems to be receiving more attention these days. Programming work-station systems may emphasize on-line non-procedural programming more and more. Two products that we saw offer this feature.

Operating modes. Programming work-stations have several operating modes. The most obvious is foreground versus background, which most have. In addition, these systems have various specialized operating modes, such as a full-screen editing mode, command editing mode, debugging mode, etc. In some systems, these are treated as sub-systems. If the system has been poorly designed, the user must shift between these sub-systems too often. We believe that it takes some *use* of a product, not just a demonstration, to decide whether it is easy or cumbersome to use.

Programming aids. We have already mentioned the most important programming aids: code format generation, syntax checking, sequence numbering, structured programming diagrams, conversational programming, and non-procedural programming aids.

In addition, some systems have added 'bells and whistles' aimed at the programming job. Here are a few of them: a MARK key, for marking certain lines of code for future quick access; an ACCESS key, for moving to these marks; a user-created abbreviation file, where the user assigns abbreviations for frequently-used words and the system automatically makes the substitutions; a FIELD key, to move the cursor to the next field; and a permanently split screen, plus one line dedicated to system status and another to messages.

Project management aids. We did not find too many built-in facilities to aid project management specifically. But we would expect to see more project management aids in the future.

Recovery facilities. When an entire programming department becomes dependent on system operation to perform even routine work, then recovery facilities are important. One system we saw enters all input information onto the disk following each carriage return. This protects all but the current line from being lost if the system goes down. One user we talked with creates a backup disk every morning and a backup tape every week.

Another type of recovery facility is one that helps a user recover after making an error. We saw two such facilities. One is an OOPS! key. It converts the last change made back to its original state. The other allows a user to retrieve a record that was just deleted. Such recovery procedures make these systems more 'forgiving' and 'friendly.'

Security features. Some of the users we talked with were very worried about programming work-station security features; others were not. Likewise with the systems, some have elaborate security features, others do not. File protection on the systems vary from (a) requiring a legal user identification and password in order to gain access to the system and all of its files, to (b) having several user-specified protection levels for all files and programs.

Target computers. The primary limiting factor in selecting a programming work-station system is: Which ones work with our in-house computer? Most stand-alone programming work-station systems are aimed at specific target machines; therefore, software developed on them can be compiled, tested and run on these targets only.

These then are the main features that we think prospective users of programming work-stations should evaluate.

We see the programming work-station field just beginning to emerge. The systems we saw appear to be a big move toward achieving greater programmer productivity.

In the future we expect work-station enhancements aimed at improving productivity in other phases of the software development cycle, such as the requirements analysis and design phases. With these, data processing departments will finally be providing their own staffs with computerized tools as sophisticated as the ones they develop for others.

REFERENCES

1. Dolotta, T. A., R. C. Haight, and J. R. Mashey, "The Programmer's Workbench," *The Bell System Technical Journal,* Bell Laboratories (Circulation Group, Whippany Rd., Whippany, New Jersey 07981); July/August 1978, pp. 2177-2200; price $2.00.
2. For more information on PWS write to Four-Phase Systems, Inc., 10700 North DeAnza Blvd., Cupertino, California 95014.
3. For more information on the Programmer's Workbench (PWB/UNIX), contact Software Licensing Department, Western Electric Company, P. O. Box 25000, Greensboro, North Carolina 27420.
4. For more information from Interactive Systems Corporation about their Workbench (IS/1), write to them at 1526 Cloverfield Blvd., Santa Monica, California 90404.
5. For more information on Pet/Maestro:
   a. In the United States, write to Itel Corporation, Maestro Systems Division, One Embarcadero Center, San Francisco, California 94111.
   b. In Europe, write to Softlab GmbH, 8000 Munich 81, Arabellastrasse 13, West Germany.
6. Drexhage, K. A. and R. E. Keirstead, "Software development and programmer work-stations," July 1979; for a copy, write K. A. Drexhage and Associates, 750 Welch Road, Suite 204, Palo Alto, Calif. 94304.
7. We have compiled a list of programmer work-station products and systems that we found in our research for this (and last month's) reports. For a free copy of this listing, write EDP ANALYZER.

Prepared by:
Barbara C. McNurlin
Associate Editor