TECHNIQUES FOR AVERTING PROBLEMS IN DEVELOPING TRAINER SYSTEM SOFTWARE

Carole J. Kuruma
Manager, Training Systems Department
Technology Service Corporation
Santa Monica, California 90405

ABSTRACT

Short schedules and changing requirements are common problems encountered when software is being developed for training systems. This paper explores techniques used by Technology Service Corporation (TSC) to overcome or avert such problems while developing the B-52 OAS Part Task Trainer for the Training Services Division, Keesler Air Force Base. Techniques for dealing with limited resources (time and budget) include carefully exploring, and assigning priorities to, system capabilities to determine the more important requirements; and employing a top-down approach. Planning for changing requirements calls for identifying capabilities that may change; constructing a well-documented software design with application-oriented modularity; and scheduling a design freeze, with late requirement changes incorporated after completion. The paper presents step-by-step descriptions of each technique and provides examples relating directly to the part task trainer.

INTRODUCTION

The U.S. Air Force B-52 bomber is receiving a major modification in the form of the Offensive Avionics System (OAS), scheduled for deployment in 1981. Used by the B-52 radar navigator to perform navigational and offensive weapon delivery tasks, the OAS replaces older ASQ-38 equipment and automates some previously manual functions.

A weapon system trainer (WST) is being built for the B-52 as a part of the major weapon system modification. The WST is scheduled to be completed in the 1983-1986 timeframe, which leaves a gap between deployment of the OAS and availability of the WST. Crews using the first OAS-modified B-52 will need a trainer in the interim. To meet this need, TSC, in conjunction with the Training Services Division, Keesler Air Force Base, has been contracted to design the software and configure four B-52 OAS interim trainers to be used by the Strategic Air Command (SAC) in their OAS conversion training program.

In approaching this assignment, TSC recognized that, in developing software for the interim trainer, two major problems also found in other software projects would be encountered: limited resources (short schedules and budgets) and changing requirements.

Short schedules are a function of the relation between the training device and the actual system. A training device cannot be specified until the actual system is designed, but it is needed as soon as the actual system--either a new one or a modification to the existing one--is operational, if not before then.

Meeting the schedule and cost constraints is then complicated by deciding which functions can reasonably be provided by the training system. The first choice of the users may be a trainer that gives a completely realistic simulation of the device. It is, after all, difficult to identify those skills that can be effectively trained with other methods, such as classroom instruction, low-cost training aids, or the system itself. The cost of such a trainer, however, may be beyond the available time and budget.

The second problem, requirement changes during development, commonly occurs because both the trainer and the system are being developed at the same time. Any changes to the system under development must be reflected in the trainer; and, by extension, since modifications to the system are likely, these same modifications must be supported by the training device.

This paper presents techniques that TSC has found to be effective in minimizing, and sometimes averting, the impact of these problems. These techniques are now being used to develop the B-52 OAS Part Task Trainer (PTT). Before detailing them, we describe the trainer as a point of reference.

DESCRIPTION OF THE PART TASK TRAINER

The interim trainer is a part task trainer, addressing some of the tasks, procedures, and conditions the crewmembers must handle with the OAS. The crews receiving training will already be skilled B-52 navigators. The focus of the PTT will therefore be on procedures training; specifically, those procedures unique to the new OAS. For example, the PTT must respond exactly as the OAS would to all commands (button pushes, switch activations, etc.), but only those features used directly in navigational and weapon-targeting procedures need to be displayed in the simulated radar video.

The PTT is laid out physically to provide stations for the two crewmembers required to operate the OAS, as well as a position for an instructor (Figure 1). Included in this layout is a mockup of the OAS crewstation, comprising 16 operational panels, four monochromatic display monitors, and two trackballs. The instructor's position is equipped with a CRT console for setting up and monitoring the training sessions. The hardware configured for the part task trainer is composed of the five independent subsystems depicted in Figure 2.

Before we describe the software and the approaches to lessening the impact of major problems in developing it, an overview of what is actually simulated by the software is necessary.

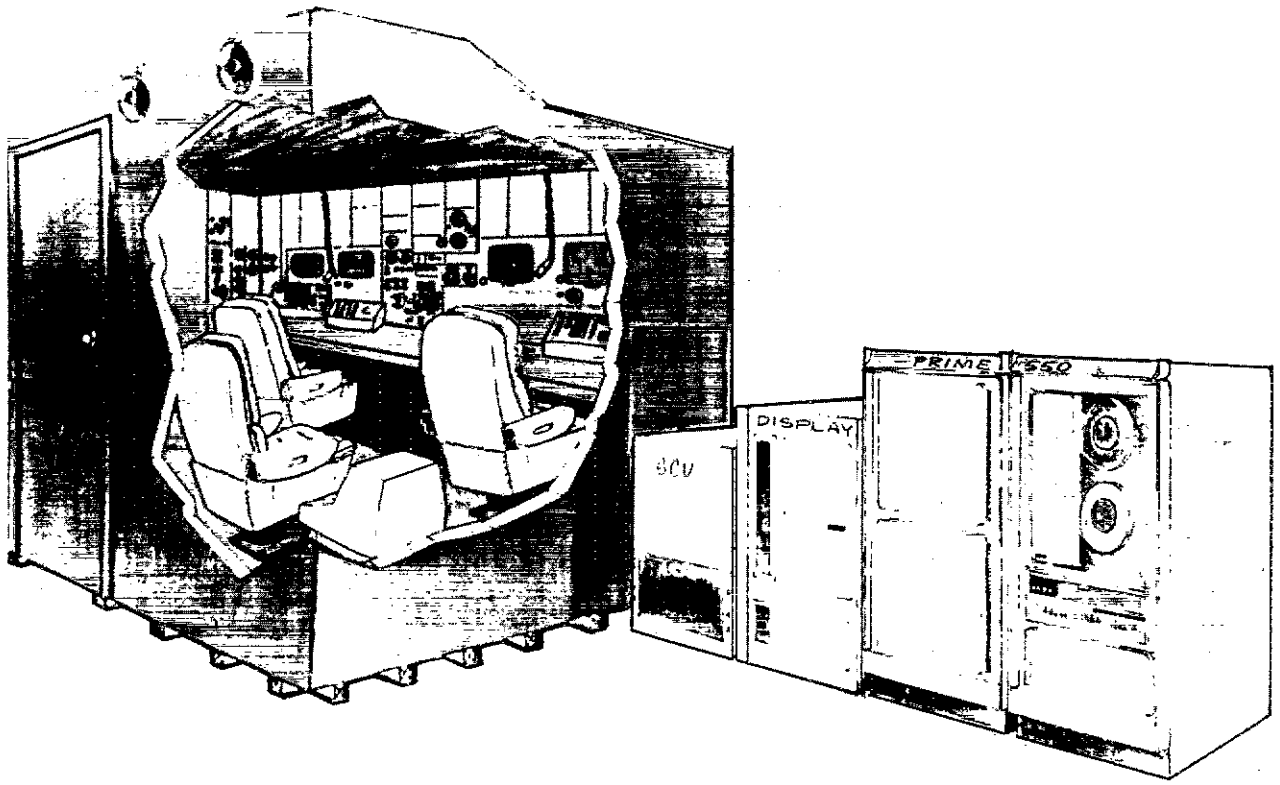

Figure 1. B-52G/H OAS Part Task Trainer

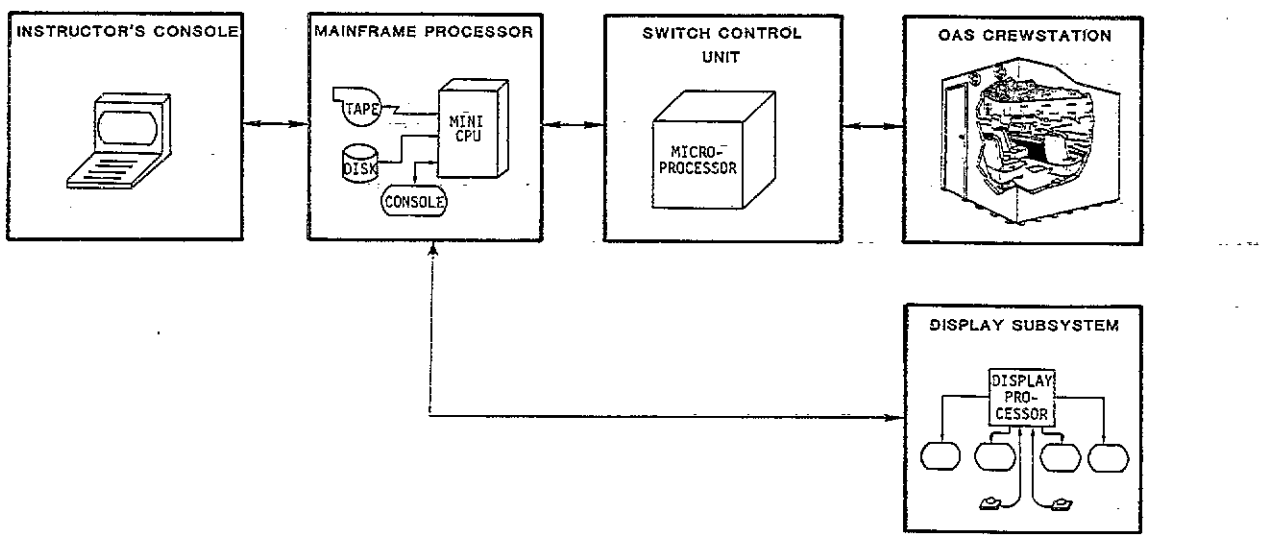

Figure 2. OAS Part Task Trainer Block Diagram

The OAS will automatically navigate the B-52 according to a predefined flight plan. The crewmembers will be able to monitor, update, and override this automatic navigation system, using the OAS equipment and a variety of navigational procedures supported by the PTT. The trainer will also support OAS procedures to perform in-flight refueling; preparation and delivery of Air-Launched Cruise Missiles (ALCMs), Short Range Attack Missiles (SRAMs), and conventional bombs; and backup procedures used when certain OAS failures occur.

As envisioned in the PTT, the instructor will be able to select a flight plan from a set of canned scenarios or modify any canned scenario to generate a new flight plan. The canned scenarios are representative of actual training missions, and scenarios of up to five hours of flight time can be generated within a region covering the Western United States. Sufficient terrain features, navigational fixpoints, and target areas are included within this region to define at least 100 different realistic scenarios. Navigational charts are used to generate synthetic radar imagery for display of these features, fixpoints, and target areas.

The instructor will also have the capability to inject faults and malfunctions at any time during the training session. With the ability to steer the aircraft off the flight plan and to accept corrections from the OAS crewmembers for recovery, the instructor will be acting as the B-52 pilot. He will also monitor crewmembers' actions and will be able to freeze the training session at any time, give additional instruction, and then resume the session.

As this brief description indicates, the PTT is a complex training device. This complexity can be further increased when the development of software for such a device is constrained by brief schedules and changing requirements. In the following sections, we present a collection of methods that we have found beneficial in a number of applications. We first discuss methods for reducing and overcoming schedule problems, and then techniques that facilitate incorporation of requirement changes.

SOME SOLUTIONS TO TIME AND BUDGET PROBLEMS

As mentioned in the Introduction, completion of the training device must coincide with or precede that of the weapon system, often resulting in short schedules. In addition, schedule slips may occur, owing to inaccurate estimates and the vicissitudes of daily life: illness, employee turnover, machine failures, and requirement changes. Specific methods TSC uses to anticipate potential schedule problems and to avoid cost overruns include carefully exploring, and assigning priorities to, system capabilities to determine the most important training functions; and employing a top-down approach so that a very limited, skeleton system that performs some of the required functions is developed first, followed by versions that successively add more capabilities until the system is complete. These two techniques are actually interconnecting, with assigning of priorities to different functions contributing to the definition of development stages or versions in the top-down approach. Careful choice of programming language and computer system, and incorporation of a programming design language also contribute to averting schedule problems and are discussed at the end of this section.

Assigning Priority

The system specification usually details the functional requirements; these functions, however, can be further ranked by the user into such categories as:

1. Necessary for training.
2. Important function without which the system will be marginally useful.
3. Important function definitely desired by the user but which could be taught, if need be, using another medium.
4. Nonessential or desirable function.

The user's initial ranking of system functions may place all functions in Category 1, and convincing the user of the importance of ranking them may be difficult (or even impossible) until schedule slips occur. It is hoped, however, that discussions with the user will be fruitful for identifying the more critical training functions. This ranking can then be used in top-down development as part of the process of identifying the successive versions.

Top-Down Development

In top-down software development, described by Yourdon, (1) the high-level design of the system is followed by implementation of a barebones system that performs some of the required functions, followed by versions that successively add more capabilities until the system is complete.

By contrast, in the classicial, bottom-up approach, the entire system is designed, coded, debugged, and then integrated. If a scheduling problem occurs, the developer and user may find, when the deadline arrives, that although 100 percent of the code is written, nothing works. The same schedule slip with the top-down approach will find a working version that may provide 75 percent of the total system functions. And although the user will not be satisfied with 75 percent of a system, that 75 percent will be more acceptable than 50,000 lines of code that are useless because they have not been integrated.

Furthermore, a user is more likely to have confidence in your ability to finish the job if he can see a working version of the partial system. For example, Version 1 of the PTT, described below, was valuable because we were able to demonstrate it to the customer, which was more effective than telling him 10,000 lines of code had been written. And, if the versions are carefully defined, the customer may be able to start training with the current version while the training device is being completed.

The most important functions, decided by the priority ranking discussed above, are scheduled for implementation in the early versions, the less important, in the later. This scheduling of functions in versions is a compromise between the logical steps required to develop the software and

the desire to provide intermediate working versions that are useful for training. For instance, the radar position fix procedure may be the most important training function; however, since it requires panel inputs, radar graphics, trackball control of the crosshair, simulation of aircraft flight and navigation systems, etc., intermediate versions are defined to develop the basic building blocks, and then the position fix procedure is scheduled for implementation in the next version.

On the PTT project, the users (SAC) were very cooperative in ranking the priority of training funcions. From this ranking, we defined seven versions (Table 1) for the top-down development of the PTT. The first three versions provide a logical development of the basic functions of the system required to support the operational procedures scheduled in Versions 4 through 7. Versions 1 through 4 have been implemented, and Version 5 is under way. The estimated time to complete each version ranges from four to nine weeks.

Version 1 may appear so basic as to be trivial. Its completion, however, was a significant event because, for one, this version required that two of the four system interfaces work: input received from the instructor's console, and output generated to the display subsystem. Yourdon discusses in detail the advantages of top-down development in testing major interfaces early in the development cycle.(1) Often these interfaces are where problems occur, and such problems are usually the most difficult to correct. Second, in this first version, although only three of 12 tasks are implemented to any degree, these three tasks are scheduled and communicate with each other, exercising a majority of the system executive routines that handle task interfacing, another area where significant problems often occur.

Version 2 added considerably more capabilities: all subsystem interfaces were exercised; inputs from one switch panel were processed by the software, allowing evaluation of response times; and most alphanumeric display formats and static radar video were displayed. Version 4 should be adequate for training the first B-52 crews, because ALCM and SRAM procedures training is not required for them; and Version 5 is expected to provide 90 percent of the necessary training functions.

TABLE 1. OAS PART TASK TRAINER VERSIONS

Version 1

- Instructor starts system in Run mode
- Aircraft flies in a straight line
- Display prime mission data, left-hand-side data

Version 2

- Instructor maneuver aircraft commands
- Aircraft flies default scenario
- Crewmember inputs from Integrated Key Boards (IKBs):
  - Select MFD
  - Select Format
  - Select Menu
  - FLY TO Command
- Display:
  - Static radar video
  - Static alphanumeric data

Version 3

- Instructor select/preview command
- Instructor's real-time parameters display
- Navigation errors modeled
- Crewmember inputs:
  - Remaining IKB functions
  - Radar Navigator's Management Panel
- Display:
  - Dynamic radar video (default format)
  - Dynamic alphanumeric data

Version 4

- Procedures:
  - Bomb run
  - Auto fixpoint sequencing
- Crewmember inputs:
  - Bomb panels
  - Special weapons panels
- Display:
  - Crosshair and residuals
  - All radar video formats

Version 5

- Procedures:
  - ALCM weapon procedures
  - SRAM weapon procedures
  - High altitude calibration
  - Radar position fix
  - OAS initialization
- Crewmember inputs:
  - Weapons Control Panel

Version 6

- Instructor commands:
  - Fault
  - Wind
  - Alternate nav heading error
  - Missile all/none status
  - Freeze/resume
- Procedures:
  - Alternate true heading calibration
  - OAS bus failure
  - Panel failures
- Crewmember inputs:
  - Remaining panels

Version 7

- Instructor capabilities:
  - Edit/save scenario
  - Post-run mode
- Procedures:
  - Point parallel rendezvous
  - Alternate bomb run
  - Terrain correlation fix
  - Low altitude calibration
- Display altitude ribbon

To show how the top-down development relates to the actual code written, Table 2 tabulates the estimated effort to complete each task and library of routines for each version. It shows that, in Version 1, a major portion of code in the library routines for error handling, data passing, input, output, etc., was developed and exercised to support a very small amount of task code. By Version 2, nearly all library routines were operational. This information is also beneficial in explaining to the customer how much effort goes into a Version 1 to produce what may be a small subset of visible functional capabilities.

TSC has found that, using top-down development, the high-level design for the system should be completed before any versions can be implemented. That is, the system executive routines which perform such functions as input/output, data base control, and intertask communication must be defined, as well as each task and data base. These definitions should include the functions of each entity, and all inputs and outputs at the functional level. For example, the task that provides the instructor's display should identify the current aircraft parameters of speed, heading, and altitude as inputs from a named data base, but would not have to specify the format of the data base. Given this high-level design, implementation of each version can proceed and the developers can be assured that no major problems will be discovered in Version 5 that could, for instance, necessitate a redesign of code developed for Version 1.

Top-down development also makes estimating easier, as well as having other advantages. The software staff prefers it because, instead of one long cycle of design, coding, and integration, the project is segmented into shorter cycles of design, coding and integration for each version. The completion of each version results in the software team having a feeling of accomplishment and enthusiasm to tackle the next version. Having several such cycles makes estimating the time to complete the remaining versions easier. By contrast, in the bottom-up approach, knowing how long it took to design and write all the code does not help to estimate how long it will take to integrate.

Programming Language and Computer System Choice

Short schedules also encourage careful choice of programming language and computer system and support software. Coding time is significantly reduced when high-level languages are used instead of assembly language. Any inefficiencies in program size caused by the high-level language can usually be offset by purchasing more memory. As for inefficiencies in execution time caused by the high-level language, those portions of code detected as causing timing problems can be rewritten in assembly language.

Using a language with which the programmers are already familiar also helps. In the B-52 OAS PTT, the IFTRAN language was used. IFTRAN is a structured FORTRAN language used by TSC for nearly all programming projects.

The minicomputer selected for the PTT is compatible with TSC's in-house system; therefore, the PTT project was able to draw upon a pool of programmers experienced with the system and a library of routines and software tools. The vendor-supplied operating system also provides many of the capabilities required for real-time systems: multitasking, semaphores, priority levels, mapped I/O, etc., thus minimizing the number of executive routines to be generated. The vendor-supplied software includes capabilities to facilitate software development: timesharing with virtual memory management that allows several programmers to develop and test code simultaneously,

TABLE 2. PERCENTAGE OF SOFTWARE DEVELOPED FOR EACH MAJOR MODULE, BY VERSION

VERSION	1	2	3	4	5	6	7
TASKS:							
COM	5	30	40	50	60	70	100
INO			70	70	70	70	100
CRW		80	90	90	100	100	100
RIK		40	50	60	70	80	100
NIK		40	50	60	70	80	100
RNM			20	40	80	90	100
WPN				15	70	90	100
AIR	5	30	40	80	90	100	100
MFD		20	50	60	80	90	100
PMD	30	40	40	80	90	90	100
XHR			5	50	80	90	100
RDR		30	40	75	90	100	100
LIBRARIES:							
IO LIB	30	70	90	100	100	100	100
EXEC LIB	50	90	100	100	100	100	100
DB LIB	80	100	100	100	100	100	100
Q LIB		100	100	100	100	100	100
NAV LIB	80	90	100	100	100	100	100
ERR LIB	60	75	75	100	100	100	100

and a source-level debugger that allows programmers to debug on-line in the high-level language.

Another suggestion for speeding up progress in software development is to have project management intervene when design discussions drag on. When the programming staff is undecided over alternative methods and no outstanding risks are identified, the chief designer must pick one method and continue. The method that is most straightforward to implement should be the one selected.

Program Design Language

Finally, it is important that development standards not be abandoned because the schedule is short. For instance, at TSC, the first step in software development is to express the design in PDL (program design language). PDL is an English language description of the design with a few structured programming keywords such as IF, ORIF, ELSE, REPEAT, WHILE. Expressed in PDL, the design is structured, machine-independent, and understandable by nonprogrammers, such as the user. Figure 3 is a sample PDL listing of a routine that controls cursor movement on a menu-driven display.

The entire software team reviews the PDL to ensure that the PDL is understandable by everyone, to detect errors and omissions, and to suggest improvements in the design. Once the design is approved, code is generated and also reviewed. Code is inserted in the same source file with the PDL, and preprocessors allow the listing of PDL only, code only, or code with PDL inserted as comments.

Use of PDL and reviews significantly shortens the integration and documentation phases of the software process in the following ways. First, PDL design reviews eliminate many of the errors that are normally not detected until software integration. Second, reviews of PDL and code ensure that each person knows enough about all the software to detect and correct many errors quickly without involving other team members. Third, software debugging is easier with PDL embedded as explanatory comments in the code. Fourth, keeping PDL and code together ensures that coding changes are also reflected in the PDL. Thus, at project completion, PDL listings can be used as final program documentation. Finally, the requirement that the software design be expressed in PDL ensures that the design is documented--and not in a disorderly set of notes or stored in a programmer's head. Thus, if the programmer becomes ill or leaves the project, the disruption is minimized because another programmer can get "up to speed" more easily.

SOME SOLUTIONS TO THE PROBLEM OF CHANGING REQUIREMENTS

Requirement changes during development of a trainer device are unavoidable when the system simulated by the trainer is undergoing simultaneous development or modification. In such a situation, a freeze should be invoked on the trainer design so development can proceed without upheavals and delays caused by changing specifications. Following completion of the software, an update phase should be planned to allow incorporation of backlogged change requests.

On the PTT project, the OAS was being built while TSC, with considerable help from SAC, was writing the functional specifications for the trainer. This task required an understanding of how the OAS would work in order to define how the PTT should support the procedures identified as training requirements. Difficulty in obtaining and understanding existing OAS documentation and coping with changes to the OAS resulted in a significant schedule slip. Working with these specification changes had one benefit: the software team recognized the need for a flexible software design to accommodate the inevitable changes in the future and obtained a good understanding of where changes might occur in the OAS.

Even when the actual system is stable throughout development of the trainer, future changes to the weapon system which must be reflected in the trainer are likely. Accepting the fact that changes are unavoidable, the trainer developers should be encouraged to provide flexibility for future changes. Design tradeoff decisions should favor the straightforward, easily modifiable approach over a more efficient method requiring a complete redesign if one of the requirements changes. As an example, one technique used by TSC is to provide many of the system parameters in separate data files that can be easily changed without affecting any of the code which uses this data.

Some of the techniques suggested for helping to meet short schedules also facilitate incorporating requirement changes: producing PDL ensures that the software design is documented, making it easier to see the impact of a change; using a high-level language and PDL as comments in the code makes the code more understandable, facilitating coding changes; and reviewing PDL and code results in a more flexible design and code to accommodate future changes.

SUMMARY

We have discussed why shortened schedules and changing requirements are often associated with the development of training systems. These conditions increase the probability of schedule slips and cost overruns or delivery of an unacceptable training device if the development plan does not adequately provide means of dealing with them. Several techniques used by TSC to minimize the impact of problems caused by schedule slips and requirement changes were presented. These techniques are being applied to the B-52 OAS Part Task Trainer, which was briefly described.

A major technique for dealing with potential schedule problems is top-down development. As described with specific examples from the PTT, top-down development entails the implementation of successive versions of the trainer so that, if delays occur, a working version of the partial system is available on the original deadline while development of the complete device continues. An important part of the definition of versions for top-down development is ranking the priority of training requirements to schedule the more critical functions in the earlier versions, maximizing usefulness of the working versions while the trainer is completed. Program design language as well as design and code reviews are other

```
      TITLE BUFFER KEYBOARD INPUT (BKI)                                  8:55:13  06/22/79
C
CD    ********************************************************************************
CD    NAME:     CURSOR TAB                      TASK:  BKI
CD    PURPOSE:  POSITION THE CURSOR ON THE MENU IN ACCORDANCE WITH THE
CD              CURSOR POSITION ENTERED BY THE USER.
CD    METHOD:   A CIRCULAR SCHEME IS EMPLOYED.  THE CURSOR ALWAYS CIRCLES
CD              AROUND IN THE SAME COLUMN OR THE SAME ROW.  WHEN AT THE
CD              BOTTOM OF A COLUMN, A DOWN TAB CAUSES THE CURSOR TO
CD              'CIRCLE' TO THE TOP OF THE SAME COLUMN.  WHEN AT THE
CD              TOP OF A COLUMN, AN UP TAB CAUSES THE CURSOR TO 'CIRCLE'
CD              TO THE BOTTOM OF A COLUMN.  THE LEFT
CD              AND RIGHT TABS HAVE THE SAME EFFECT.  THAT IS IF THERE
CD              EXISTS ANOTHER COLUMN (POSSIBLE ONLY IN COMMAND SELECT)
CD              THEN LEFT OR RIGHT POSITIONS THE CURSOR IN THE NEXT
CD              COLUMN.
CD    INPUT PARAMETERS:
CD              CURRENT INPUT STATE (FOR BKI)
CD              COMMAND ID
CD              PARAMETER ID
CD              CURSOR KEYSTROKE (FROM MOC KEYBOARD)
CD    OUTPUT PARAMETERS:
CD              COMMAND ID (UPDATED)
CD              PARAMETER ID (UPDATED)
CD              COMMANDS FOR THE MOC TO POSITION CURSOR
CD    DATA BASE USAGE:
CD              NONE
CD    INVOKING METHOD:
CD              INVOKE CURSOR TAB
CD    INVOKED BY:
CD              COMMAND SELECT KEYSTROKE (TO POSITION CURSOR)
CD              PARAMETER SELECT KEYSTROKE (TO POSITION CURSOR)
CD    BLOCKS INVOKED:
CD              NONE
CD    ********************************************************************************
              BLOCK CURSOR TAB
1             .   IF DOWN CURSOR KEYSTROKE
2             .   .   IF AT THE BOTTOM OF A COLUMN
3             .   .   .   PUT CMNDS IN MOC BUFFER TO POSITION CURSOR AT TOP OF COLUMN
2             .   .   ELSE : NOT AT THE BOTTOM OF A MENU
3             .   .   .   PUT CMNDS IN MOC BUFFER TO POSITION CURSOR AT NEXT ROW DOWN
2             .   .   ENDIF
1             .   ORIF UP CURSOR KEYSTROKE
2             .   .   IF CURSOR IS AT THE TOP OF A COLUMN
3             .   .   .   PUT CMNDS IN MOC BUFFER TO POSITION CURSOR AT BOTTOM OF COLUMN
2             .   .   ELSE : NOT AT THE TOP OF A MENU
3             .   .   .   PUT CMNDS IN MOC BUFFER TO POSITION CURSOR AT NEXT ROW UP
2             .   .   ENDIF
1             .   ORIF KYSTRK IS LEFT/RIGHT AND NOT IN PARAMETER SELECT INPUT STATE
2             .   .   IF THERE ARE TWO COLUMNS & MENU COMMANDS IN THE NEXT COLUMN
3             .   .   .   PUT CMNDS IN MOC BUFFER TO POSITION CURSOR INTO NEXT COLUMN
2             .   .   ENDIF
1             .   ORIF KEYSTROKE IS A HOME KEYSTROKE
2             .   .   PUT CMNDS IN MOC BUFFER TO POSITION CURSOR AT TOP OF MENU
1             .   ENDIF
1             .   IF CURRENT INPUT STATE IS COMMAND SELECT
2             .   .   MODIFY COMMAND ID
1             .   ELSE : CURRENT INPUT STATE IS PARAMETER SELECT
2             .   .   MODIFY PARAMETER ID
1             .   ENDIF
              ENDBLOCK : CURSOR TAB

      CD      ********************************************************************************
```

Figure 3. Program Design Language (PDL)

techniques that help to avoid schedule slips and facilitate requirement changes.

Programmer familiarity, ease of use, and availability of required operating system functions and software tools are factors which should be considered in selecting the computer for the training device in order to help meet shortened schedules.

REFERENCE

1. Yourdon, E., Managing the Structured Techniques, Yourdon Press, New York, 1979.

ABOUT THE AUTHOR

Ms. Kuruma is the manager of Technology Service Corporation's Training Systems Department. She has over 12 years' experience in project management and real-time software development.