APPLICATION OF SOFTWARE DESIGN METHODOLOGY FOR SMALL MAINTENANCE TRAINERS

MICHAEL C. DASHOW AND MICHAEL A. KOGAN
Grumman Aerospace Corp.

PURPOSE

The prime purpose of Simulated Avionics Maintenance Trainer (SAMT) Common Core Program was, and still is, to design and develop the hardware and software designs and techniques which will allow us to build and deliver training equipment that is cost effective and very responsive to short delivery schedule restrictions. Common hardware and software modules are to be developed which will supply our company with a menu of technology available for the SAMTS of the near future. It is anticipated that this will be a continuing effort so that we can update our capabilities as new requirements come over the horizon.

BACKGROUND

Before committing to an approach to the SAMT Common Core, Grumman carefully studied the future requirements for maintenance trainers of the near future by evaluating future procurements and doing extensive in-house studies. Based on our knowledge of several possible trainers, we ascertained that the following characteristics would be included in new maintenance trainers:

- Multi-Student station operation
- Incorporation of many of the Computer Based Education (CBE) features
- Controlling a variety of visual projection devices
- Controlling various types of interactive CRT terminals
- Capability of user to modify lessons readily without requiring programming knowledge.

Based on the preceding criteria, the decision was made to use the emerging 16 bit microprocessor for all Common Core development effort because of the following characteristics:

- Ability to directly address large size memory (1 Megbyte and up) which is required for CBE applications
- Readily adaptable to multiprocessing, the requirement imposed by multi-student station operation.

Additionally, this size of microcomputer system fit in very well with the capabilities we had developed on other size microcomputers and minicomputers.

A survey was made of the available 16 bit microprocessors; those suppliers that were given serious consideration are shown in Table 1.

LONG RANGE GOALS

This program has as its goal the generation of software modules, having maximum portability, for a family of microcomputers and minicomputers which, together with a group of circuit design modules, will allow a "mix and match" selection to cover most simulated electronics maintenance trainer requirements.

Specific long range goals include:

- 16 bit microprocessor based instructor/operator station to interface with multi-student stations
- Software module to control a 120 image, or greater, microfiche/audio system with random access
- Software module to control a random access video system
- Software module to handle dumb and smart terminals
- Software module to handle standard peripherals
- Software module to handle student scoring and evaluation
- Software module(s) to demonstrate CAI capabilities
- Signal generation and display using software and hardware modules developed from on-going projects and development of new techniques
- Design of hardware modules to represent typical system and support equipment indicators and controls including:
  - Single and multiposition switches
  - Analog inputs (pots, control stick, frequency selection, etc.)
  - Digital displays

TABLE 1 16 BIT MICROPROCESSOR CHARACTERISTICS

MFG/CHIP	AVAILABILITY	SECOND SOURCE	DESCRIPTION	STATUS	SPECIFICATION	SUPPORT	
						HARDWARE	SOFTWARE
DATA GENERAL MN601 (MICRO NOVA)	NOW	NONE	IMPLEMENTATION OF NOVA 3, INCLUDING MEMORY AND SIMILAR I/O.	LOWER RISK CHOICE FROM STANDPOINT OF SUPPORT SOFTWARE, HOWEVER PARTS REMAIN RELATIVELY EXPENSIVE. NO DUPLICATION FROM OTHER MFG.	EXECUTES NOVA 3 INSTRUCTIONS AND APPROXIMATES NOVA 3 I/O. MAX OF 32KB MEMORY.	CAN USE REGULAR NOVA 3 MINICOMPUTER AS HOST FOR SOFTWARE DEVELOPMENT OR MICRO NOVA. HAVE PACKAGED I/O, AND MEMORY BOARDS.	WIDE RANGE OF SOFTWARE EXISTS. DOS, RTOS, FORTRAN IV, BASIC, ETC.
FAIRCHILD 9440/9445	*NOW 9440* 1979 9445	*NONE*	*CIRCUITRY IMPLE-*MENTS NOVA 1200 AND NOVA 3 MINI COMPUTERS.	*DATA GENERAL HAS* INSTITUTED LEGAL ACTION AGAINST FAIRCHILD FOR HARDWARE AND SOFTWARE INFRINGEMENTS. DG WON A SIMILAR CASE AGAINST DIGITAL COMPUTER CONTROLS SEVERAL YEARS AGO.	CURRENT 9440 EXECUTES ALL INSTRUCTIONS OF OLDER NOVA 1200. PLANNED 9445 WILL EXECUTE INSTRUCTIONS OF NOVA 3.	EVALUATION BOARD: SPARK 16 PC BOARD MAY BE USED AS OEM BOARD OR STAND ALONE COMPUTER. INCLUDES CPU, 4K RAM, 2K PROM INCL. MONITOR, MEMORY CONTROL WITH DMA, LOGIC FOR SERIAL PORT. REQUIRES TTY OR CRT.	SOFTWARE AVAILABLE THROUGH BASIC AND MACRO-ASSEMBLER. IN NEXT SIX MONTHS, DOS, IDOS, REALTIME EXEC, FORTRAN, AND PASCAL.
INTEL 8086	NOW	MOSTEK	INTEL OBJECTIVE IS TO OFFER MACHINE THAT MATCHES PERFORMANCE OF MID-RANGE MINI-COMPUTERS, BUT RETAINS SOME UPWARD COMPATIBILITY WITH 8080/8085.	USER BENEFITS FROM COMPANY'S POSITION AS RELIABLE, AGRESSIVE SUPPLIER WITH BROAD BASE OF SUPPORT, I.E., COMPATIBLE MEMORIES, BUS DRIVES, SUPPORT CHIPS, AND DEVELOPMENT TOOLS.	*CPU CAN REACH 1* MEGABYTE MEMORY SPACE. MULTIBUS ARRANGEMENT IS READILY ADAPTABLE TO MULTIPROCESSING.	*INTELLEC MODEL 230* DEVELOPMENT SYSTEM ICE 86 FOR SOFTWARE AND HARDWARE DEVELOPMENT. SDK-86 SINGLE BOARD MC.	*MODEL 230 ISIS-II* DOS, ASM-86 ASSEMBLER, CONV-86 (8080 TO 8086 TRANSLATOR) PL/M-86 COMPILER, ICE-86 SOFTWARE SDK-86 FIRMWARE MONITOR/DEBUGGER.
MOTOROLA 68000	1979 AVAILABILITY INDEFINITE	NONE -- POSSIBLY SAME FIRMS THAT SECOND SOURCE 68000	MOST AMBITIOUS NMOS μP DESIGN PRESENTED, HAS 32 BIT WIDE CPU REGISTERS, DATA PATHS AND ALU; OUTPUTS 24 BIT ADDRESS TO DIRECTLY REACH 8 MEG.	SUPPLIER PUSHING STATE OF ART BY PACKING 75000 ACTIVE ELEMENTS ON A 64 PIN PACKAGE.	LARGE COMPUTER ARCHITECTURE, 24 BIT TOTAL ADDRESS LINES TO ADDRESS 16M LOCATIONS. 16 BIT WIDE EXTERNAL DATA BUS EXPANDS TO 32 BIT INTERNAL BUS, (8) 32 BIT DATA REGISTERS, (8) 32 BIT ADDRESS REGISTERS, ALU IS 32 BITS WIDE.	MOTOROLA PLANS TO USE EXORCISER DEVELOPMENT SYSTEM TO RUN DEVELOPMENT SOFTWARE AND CONTROL PROTOTYPES VIA ICE. ABOVE AREAS ARE VAGUE AT MOMENT	PROBABLY DEVELOPING CROSS ASSEMBLER FOR 68000 THAT WILL RUN ON EXORCISER STATION.

TABLE 1 16 BIT MICROPROCESSOR CHARACTERISTICS (Cont)

MFG/CHIP	AVAILABILITY	SECOND SOURCE	DESCRIPTION	STATUS	SPECIFICATION	SUPPORT	
						HARDWARE	SOFTWARE
TEXAS INSTRUMENTS 9900-9980/81 SBP 9900A	NOW FOR ALL THREE MODELS	AMI-9900 AND 9980/81 SMC – 9980 NONE FOR SBP 9900A	EXTREME EXAMPLES OF MEMORY TO MEMORY ARCHITECTURE (INCLUDING CPU REGISTERS). MP SUITABLE FOR MULTIPROCESSING AND RAPID CONTEXT SWITCHING UPON INTERRUPT OR SUBROUTINE CALL. DEVICES ARE PART OF BROAD LINE.	TI HAS INVESTED HEAVILY IN THIS FAMILY, BUT HAS NOT GAINED THE LEADING MARKET. 1979 PROBABLY CRITICAL YEAR FOR 9900 WITH ADVENT OF OTHER 16 BIT μC.	UNIFIED MEMORY ARCHITECTURE IN WHICH WORKSPACE POINTER IN CPU DEFINES WHICH 32 BYTE SECTION OF RAM WILL BE USED FOR CPU WORKING REGISTERS; 64 PIN PACKAGE CAN ACCESS 32K, 16 BIT WORDS EXTERNALLY FOR 9900; 9980, AND 9981 IN 40 PIN DIP AND CAN ONLY ACCESS 8K 16 BIT WORDS.	DEVELOPMENT SYSTEM: AMPL 990 MINI BASED STATION WITH CRT, FLOPPY OR HARD DISC; SUPPORTS MULTIPLE USERS. EMULATOR BOARDS TO INTERFACE TO USER PROTOTYPES; BOARD LEVEL ASSEMBLIES.	FOR AMPL: HIGHER LEVEL LANGUAGES INCLUDE C030L, FORTRAN, BASIC PASCAL AND AMPL. FROM PROGRAMMING AND ICE. TIME-SHARING AVAILABLE ON GE, NCSS AND TYMSHARE.
ZILOG Z8000	BEST GUESS IS 2ND QUARTER FOR Z8000; SECOND HALF 1979 FOR MEMORY MANAGEMENT CHIP.	AMD, LICENSED	HAS ARCHITECTURE OF LARGE MINI (PDP-11) AND MAIN FRAME (IBM 360/370) MACHINES. COMES IN 40 PIN PACKAGE FOR 64K MEMORY AND 48 PIN PACKAGE FOR ADDRESSING UP TO 8M BYTE MEMORY. CLAIMED TO OUTPERFORM PDP-11/45.	AMBITIOUS CHIP THAT ZILOG HAS HAD DIFFICULTY IN PRODUCING. DELAY OF ALMOST ONE YEAR. WITH THIS MP, SUPPLIER IS DEPARTING FROM 8080 ARCHITECTURE. Z8000 WILL HAVE OWN LINE OF I/O SUPPORT CHIPS.	DIRECTLY ADDRESSABLE MEMORY SPACE OCCUPIES 64K LOCATIONS BUT CAN BE EXPANDED TO 8M LOCATION BY SEGMENT SELECT. 100 BASIC INSTRUCTIONS WITH 410 COMBINATIONS. EIGHT LARGE COMPUTER STYLE ADDRESSING MODES PLUS UNUSUALLY LARGE REPERTOIRE OF BLOCK AND STRING OPERATIONS. READILY ADAPTABLE TO MULTIPROCESSOR OPERATION. REQUIRES SEPARATE MEMORY MANAGEMENT UNIT FOR 8M MEMORY.	ZDS Z80 BASED DEVELOPMENT SEPTEMS WILL RUN Z8000 CROSS SOFTWARE. Z8000 SYSTEM ANALYZER WILL PERMIT ZDS TO COMMUNICATE WITH Z8000 PROTOTYPE.	ZILOG EXPECTS TO SUPPORT Z8000 WITH TRANSLATORS FOR PLZ, BASIC, COBAL, AND FORTRAN. THESE WILL PERMIT CONVERSION OF Z80 CODE TO Z8000 CODE SINCE Z8000 SET IS SUPERSET OF Z80.

- Counters (electrical, mechanical)
- Analog outputs (meters)
- Audio output

- Software module to handle all input/output operations of the simulator
- Software module to handle all input/output operations of the simulator
- Software module for initialization of the system
- Develop maintenance algorithm and software module for a demonstration trainer which can be used with minor modification for a wide range of trainers.

INITIAL OBJECTIVES

The guidelines established for the initial phases of this effort were keyed to a straightforward implementation of the target concepts and methodologies. This approach limited the ambiguities in the analysis and evaluation of results. It also provided for a cost effective and expeditous production of a device to meet near term requirements. Such a device is shown in Figure 1.

Specific goals are as follows:

- Develop a "sample" student station using a 16 bit microprocessor and typical aircraft (A/C) system controls
- Interface to a display device (video disc or microfiche) at the student station
- Interface student station computer to interactive terminal
- Develop a very limited maintenance trainer program to demonstrate capability
- Determine requirements for an instructor station capable of handling multiple student stations using the same 16 bit microcomputer
- Determine impact on SAMT software of a portable language for computer aided instruction.

SAMT COMMON CORE DESIGN

HARDWARE

As stated previously, it is our intent to stress modularity in the design of the SAMT Common Core, since this is the technique that Grumman feels offers the most cost effective approach for its development efforts for maintenance trainers.

In general, trainers consist of the System Simulation and the Computing System. Most modern computers are designed and built in a modular fashion so that the unit can be customized readily to the user's requirements. This is especially true of minicomputers and microcomputers, since these configurations are sized for the "smaller"

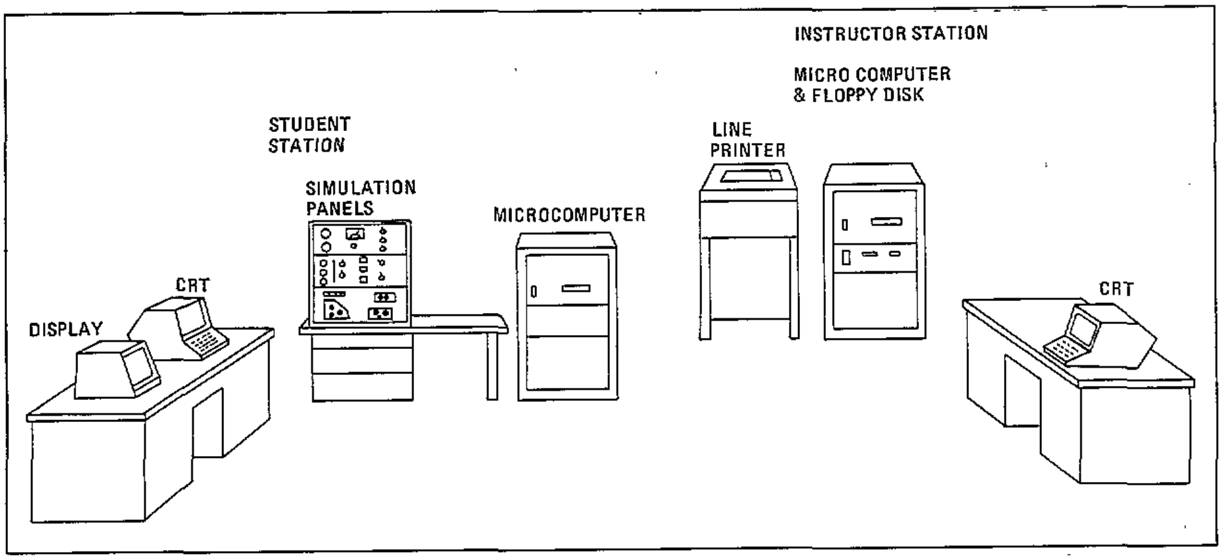


Fig. 1 SAMT Common Core

user where a myriad of different tasks are presented by the spectrum of users. Consequently, these systems must be readily adaptable, with minimum cost impact, to the various user requirements. This is accomplished with a bussed structure shown in the overview diagram of the Modular Computing System of Figure 2.

A SAMT could have any or all of the following hardware modules:

- Central Processor Unit
- N Memory Modules
- Mass Storage Unit
- Peripheral Devices
  - CRT/Keyboard
  - Teletype

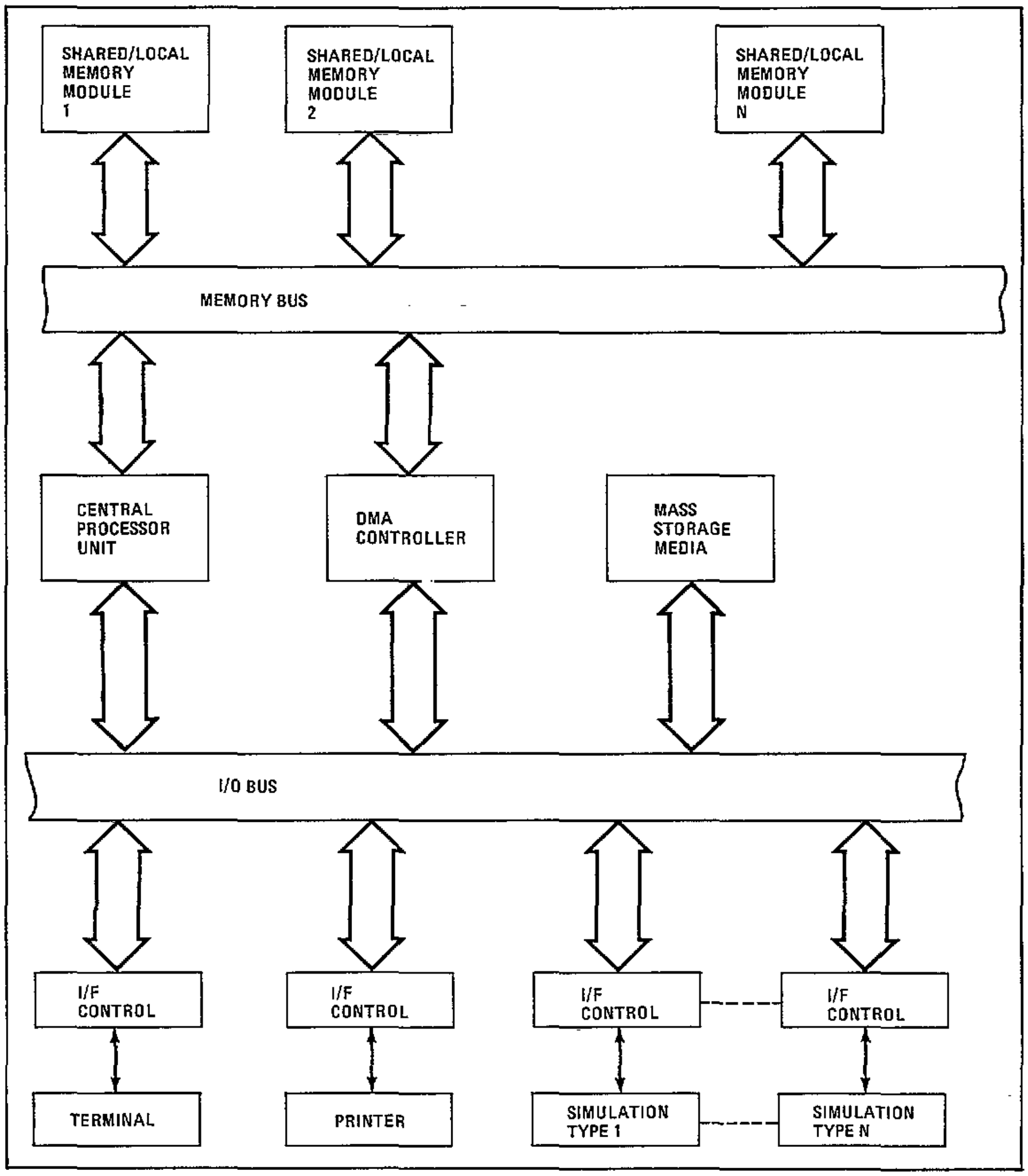


Fig. 2 Modular Computing System

- Paper Tape Punch/Reader
- Line Printer
- Character Printer

- DMA Control
- N Simulations of n devices.

A minimum configuration might be the Central Processor Unit, (1) Memory Module, and terminal, such as a CRT/Keyboard or a Teletype. One can see that the buss structure readily adapts to the addition of hardware modules limited only to the addressing capability of the chosen computer, line and/or character printers, mass storage media and the various types of simulation, as required by the particular trainer specification. In most cases, the Interface Control (I/F Control) can be obtained as a predesigned unit for standard peripheral devices from the chosen computer manufacturer.

Since the test equipment and/or devices to be simulated for maintenance trainers can be categorized to particular functions such as switches, lights, meters, oscilloscopes, test points, etc., the simulation hardware can be modularized by function and optimized for module capacity determined by the study of the SAMT requirements.

SOFTWARE

Near the end of a software development effort, the phrases, "it's about 90% complete", and "just a few more bugs", are usually the response to overall software progress. It is usually at this point that design flaws are found requiring reprogramming to fit "new" things in, additional overtime, schedule slippage, and retesting of everything. Because of these facts, it is imperative that the methodology for the generation of software be made as cost effective as possible. The techniques which assist greatly in meeting this goal are:

- Top-down program development
- Structured programming techniques
- Modular program development.

TOP-DOWN/BOTTOM-UP DEVELOPMENT

Two techniques commonly used for program development which were considered for the SAMT Common Core effort are top-down and bottom-up approaches. The bottom-up approach requires that the lowest level program modules are coded first and then tested. Each Module Under Test (MUT) requires a Special Exercise Routine (SER). The SER permits transmission of test data to the MUT and receives output from it so that intermediate results can be verified as being correct. Since SERs are not modules of the final program, they represent additional time and resources spent on an undeliverable item. At first glance, bottom-up appears to be a pyramid building scheme, whereby a level of confidence is established in the correctness of each MUT prior to going on to the next highest module. Theoretically, the total integration of all modules should be a minimal effort leading to the successful completion of the project. However, there are several reasons why this situation seldom occurs. First, each MUT can work correctly with its SER, but all MUTs may not work together. This occurs when module interfaces were interpreted differently by the programmers of each module. A second source of errors occurs when higher level module specifications are changed without making a corresponding change in a lower level module. These aspects of bottom-up development lead to a situation where final integration becomes the most difficult and time consuming part of a programming project. Because of these problems, this approach was discarded for the SAMT Common Core. These difficulties can be avoided if a top-down approach is applied.

Top-down development is another method for program design and construction. Requirements are repeatedly broken down into simpler functions, which results in a hierarchical structure of identifiable programs and modules. The highest level of control logic resides at the top of the hierarchy, while the computational or algorithmic functions reside at the lower level. The program is divided into constituent parts and these parts are broken down into their constituents. Each level of development (or breakdown) is continued until a level is reached where there is no subservient level. Levels are structured so that a lower level does not call on a higher level. Each level in a program hierarchy chart represents a function of a module. Figure 3 is the program hierarchy chart developed for the SAMT Common Core effort and Table 2 is a tabulation of module descriptions. Top-down philosophy means that higher level modules are written and integrated <u>before</u> those below them, thus resulting in continual verification of module interfaces. Consequently, if errors are found, they are localized within newly added modules. Also, debugging is facilitated and integration becomes continuous, rather than the time consuming effort of bottom-up development near the end of a programming task.

STRUCTURED PROGRAMMING TECHNIQUES

Trainer system software implements training operations and doctrine in areas susceptible to many changes of performance requirements. These changes often impact the software and need expeditious implementation. This demands that trainer system software be de-

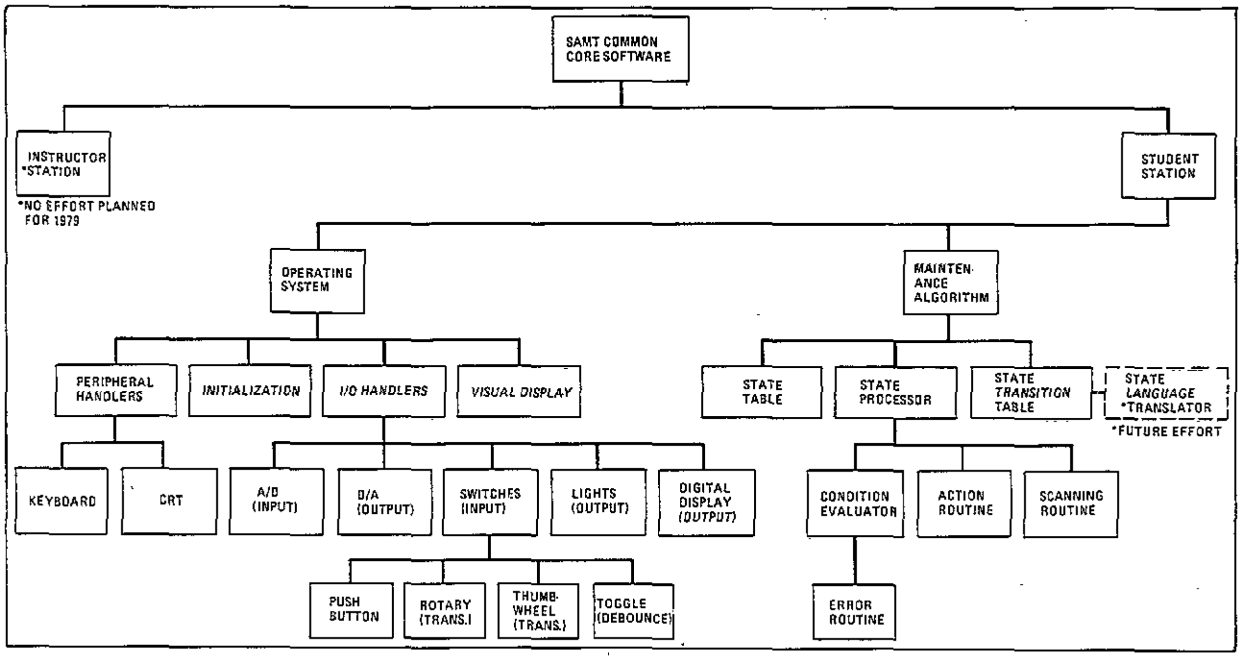


Fig. 3 SAMT Common Core Hierarchy Chart

signed to facilitate efficient change. The application of structured programming techniques to the SAMT Common Core helped meet this requirement.

Structured programming is a methodology for writing programs which are reliable, readable, maintainable, and easily modifiable by other programmers. The constructs used are all structured, having a single entry and exit point. The five basic structures (see Figure 4) used for decomposing any module are: the SEQUENCE of operations (assignment, add, ...), IF THEN OR ELSE, DO WHILE, DO UNTIL and CASE. The purpose of these structures is to produce correct programs; i.e., it actually does what it is supposed to do. By using the basic structures, the logic flow of a module will be from top to bottom. This forward direction of program logic makes a program easier to read when it must be modified. Since modules represent functions having only a single entry and exit point they represent a pattern of linear reasoning which by itself assists in writing correct code.

MODULAR PROGRAM DEVELOPMENT

The terms Module and Program have been used without definition. An accepted definition of Module is that it is a collection of logically related code having a single entry and exit point that performs a single function. Characteristics of a Module are:

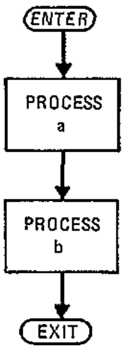
- The cause of errors can be easily isolated to a single pair of modules.
- Program development is expedited since it is easier to code, test, and debug several simple modules than one complex program.
- Modularity permits the partitioning of the program development effort into parallel tasks.
- The ability to replace modules which perform inefficiently.
- Each module should have a descriptive name, as well as a sequence number, so that the module can be uniquely identified for configuration control purposes.
- Increased comprehensibility since it is possible to study the program one module at a time.
- Independently compiled modules reduce overall compilation time and resource usage.
- Modules should not exceed more than 200 executable statements.

The Maintenance Algorithm developed for the SAMT Common Core is trainer independent; i.e., only the lesson sequences change. Modularizing the Maintenance Algorithm enabled us to establish single point access to the Operating System (O/S). This

TABLE 2 SAMT SOFTWARE MODULE DESCRIPTION

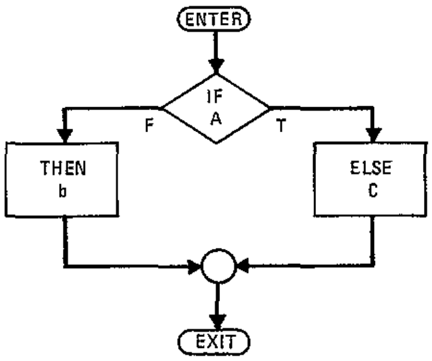
MODULE NAME	DESCRIPTION
STUDENT STATION OPERATING SYSTEM	A SET OF SOFTWARE MODULES WHICH TAKES CARE OF THE NORMAL TASKS OF THE COMPUTING SYSTEM. THE SOFTWARE IS NOT RELATED TO A PARTICULAR TRAINER CONFIGURATION, BUT RATHER IS DESIGNED TO ACCOMPLISH THE NORMAL HOUSEKEEPING ROUTINES REQUIRED FOR SYSTEM OPERATION. THE ROUTINES REQUIRED FOR THE SAMT COMMON CORE ARE DISCUSSED BELOW.
PERIPHERAL HANDLERS	THESE MODULES WILL ACCOMMODATE THE INTERFACING AND TRANSFER OF DATA TO AND FROM THE STANDARD PERIPHERALS USED WITH COMPUTING SYSTEMS. CATEGORIES OF INSTRUMENTS HANDLED ARE CHARACTER PRINTERS, LINE PRINTERS, MAGNETIC TAPE UNITS AND CRT/KEYBOARDS. THE CRT/KBD MODULE ONLY WILL BE DEVELOPED FOR THE 1979 EFFORT. TASKS INCLUDE USING THE INDUSTRY STANDARD RS232, HANDLE SERIAL DATA TRANSMISSION, ECHO OF KBD GENERATED CHARACTER THROUGH COMPUTER TO CRT, DECODING OF CHARACTERS AND MNEMONICS FOR COMMAND GENERATION.
INITIALIZATION	THE PURPOSE OF THIS MODULE IS TO INITIALIZE THE COMPUTING SYSTEM TO THE MODE OF OPERATION REQUIRED BY THE TRAINER CONFIGURATION. TASKS INCLUDE PLACING SYSTEM INTO RESET STATE, PERIPHERAL CHIPS MODE AND COMMAND SETUP, INTERVAL TIMER MODE SETUP (AT LEAST THREE TIMERS, PARALLEL I/O PORT MODE SETUP, PROGRAMMABLE INTERRUPT CONTROLLER MODE AND COMMAND SETUP, ANALOG INPUT/OUTPUT BOARD MODE AND COMMAND SETUP, AND MULTI BUS MEMORY, PARALLEL I/O PORT, AND SERIAL CHANNEL MODE AND COMMAND SETUP.
I/O HANDLERS	THESE SOFTWARE MODULES WILL BE USED TO TRANSFER DATA TO AND FROM THE VARIOUS HARDWARE MODULES IN THE TRAINER SYSTEM. DEVICES TO BE HANDLED INCLUDE A/D AND D/A CONVERSION, LAMP CONTROL, DIGITAL DISPLAY AND VARIOUS SWITCHES (PUSHBUTTON, ROTARY, THUMBWHEEL, TOGGLE). ROTARY AND THUMBWHEEL SWITCH TRANSITIONS AND TOGGLE SWITCH DEBOUNCE ROUTINES MUST BE INCLUDED. SWITCHES MUST BE SCANNED AT A REPETITION RATE SUFFICIENTLY FAST TO RECORD ANY CHANGES ENTERED BY THE TRAINEE. DIGITAL DISPLAYS MUST BE REFRESHED AT A RATE TO PREVENT ANY NOTICEABLE BLINKING OF THE ENERGIZED SEGMENTS.
VISUAL DISPLAY	THE PURPOSE OF THIS MODULE IS TO INTERFACE WITH THE SELECTED VISUAL DISPLAY. NECESSARY SIGNALS WILL HAVE TO BE DEVELOPED TO DRIVE THE DISPLAY. MODIFICATION OF STRAIGHT BINARY CODE TO DISPLAY PECULIAR CODE MAY BE A REQUIREMENT.
STUDENT STATION MAINTENANCE ALGORITHM	A GROUP OF SOFTWARE MODULES WHICH IMPLEMENTS THE TRAINING REQUIREMENTS OF THE PARTICULAR TRAINER CONFIGURATION. MAINTENANCE TRAINERS CAN BE
STUDENT STATION MAINTENANCE ALGORITHM (CONTINUED)	VISUALIZED AS SEQUENCES OF STATES, DOING A SEQUENCE OF THINGS ONE AT A TIME, AS COMMANDED BY ITS USER. THE TRAINER'S STATE IS DEFINED BY ITS CURRENT ACTIVITY. THE ALGORITHM, THEN, HAS BEEN DESIGNED TO BE TABLE DRIVEN. THE VARIOUS SOFTWARE MODULES WILL BE DESIGNED WITH THE GOAL OF BEING TRAINER INDEPENDENT WITH ONLY THE STATE TABLE, STATE TRANSITION TABLE, AND ERROR ROUTINE BEING TRAINER DEPENDENT.
STATE PROCESSOR	THIS SOFTWARE MODULE CONSISTS OF FOUR SUB-PROGRAMS, WHICH TOGETHER WITH THE STATE TABLE AND THE STATE TRANSITION TABLE (STT) PROCESS THE ACTIONS OF THE TRAINER. THE ALGORITHM HAS BEEN DESIGNED TO WORK WITH TRANSITIONS ONLY, WHICH ALLOWS US TO REDUCE THE DATA BASE REQUIREMENTS SIGNIFICANTLY. THE FOUR SUBROUTINE TASKS ARE AS FOLLOWS: **SCANNING** – CALLS THE VARIOUS I/O HANDLERS, DETERMINES IF THERE HAS BEEN A CHANGE OF STATE, AND IF SO, IDENTIFIES THE INPUT DEVICE THAT HAS CHANGED AND POSTS THE CHANGES. **CONDITION EVALUATOR** – DETERMINES IF THE CORRECT STATE CHANGE HAS OCCURRED BY A COMPARISON TO THE INDEXED STEP OF THE STT. DEPENDING ON THE RESULTS OF THE EVALUATION, EITHER OF THE FOLLOWING SUBPROGRAMS IS CALLED BY THE EVALUATOR. **ACTION** – ACTIVATES THE CORRECT DISPLAY, LAMP, METER OR ANY TRAINER REACTION TO THE STUDENT CORRECT ACTION BY CALLING THE REQUIRED I/O HANDLER. THIS IS DONE IN *CONJUNCTION WITH THE DATA STORED* AT THE INDEXED STEP OF THE STT. **ERROR** – THIS ROUTINE IS CALLED AS A RESULT OF THE CONDITION EVALUATOR DETERMINATION OF AN INCORRECT ACTION ON THE PART OF THE TRAINEE. THIS SUBPROGRAM IS TRAINER PECULIAR SINCE EACH CUSTOMER WILL NO DOUBT HAVE DIFFERENT ERROR INDICATIONS TO THEIR TRAINEE. THE COMMON CORE WILL TABULATE AND EVALUATE EACH ERROR AND THROUGH GENERATIVE COMPUTER AIDED INSTRUCTION (CAI) WILL GUIDE THE STUDENT TO AN APPROPRIATE RESPONSE. IT IS ANTICIPATED THE VARIOUS INDICATIONS WILL INCLUDE ERROR LAMPS, CRT MESSAGES PLUS AUDIBLE ALARM, AND VISUAL DISPLAYS.
STATE TABLE	THIS TABLE IS THE PRESENT STATUS OF THE VARIOUS SIMULATION DEVICES OF THE TRAINER AND IS TRAINER PECULIAR. WHEN THE LESSON IS STARTED, THE INITIAL STATE OF ALL OF THE DISCRETE INPUT (DIs) AND DISCRETE OUTPUTS (DOs) ARE POSTED INTO THE TABLE. AS THE LESSON IS STEPPED THROUGH, IT IS CHANGED TO REFLECT THE TRANSITIONS OF THE TRAINING DEVICE.
STATE TRANSITION TABLE	THIS TABLE IS TRAINER PECULIAR IN THAT IT REFLECTS THE ACTIONS REQUIRED BY THE LESSON. FOR EACH STEP OF THE LESSON EACH TRAINEE ACTION OR ACTIONS IS STORED SUCH AS, INPUT IDENTIFICATION AND TYPE. THE CORRESPONDING TRAINER REACTION IS ALSO STORED AT THE STEP NUMBER. AS THE LESSON PROGRESSES, THIS TABLE IS STEPPED TO THE CORRESPONDING TEST NUMBER.

A. SEQUENCE: Process a Process b

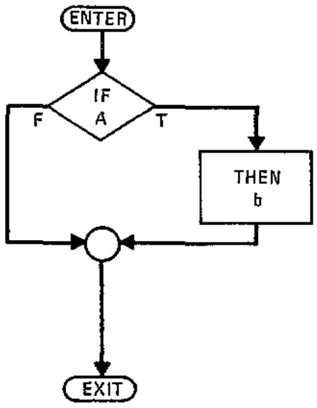


Control flows from process a to the next in sequence, process b.

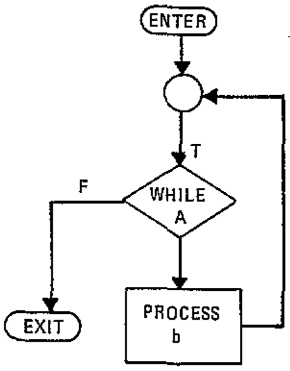
B. IF THEN ELSE: If condition A THEN b, ELSE process c.



The flow of control will return to a common point after executing either process b or c. A predicates the conditional execution. If control is to skip a process pending the condition of A, then the flow chart can be modified thusly:

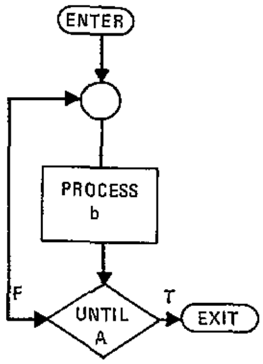


C. DO WHILE: DO WHILE condition A process b



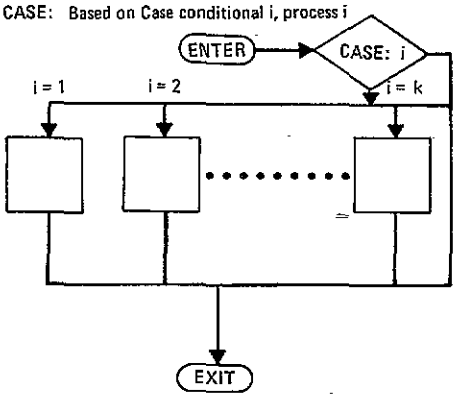
The DO WHILE *structure is a loop in which the condition A is* evaluated. If found to be true, then control is passed to process b and the condition A is evaluated again. If condition A is false, then control is passed out of the loop.

D. DO UNTIL: DO UNTIL condition A Process b



The DO UNTIL structure is similar to the DO WHILE – except that the test of condition A is performed after process B has executed. Thus *the* DO UNTIL *loop will be performed once regardless of the value of condition A.*

E. CASE: Based on Case conditional i, process i



Control is passed to process k based on the value of i.

Fig. 4 Control Structures

modularity approach was applied to the O/S, thus creating interface routines which can be easily substituted for others, depending upon the trainer performance characteristics.

TESTING

In order to conform to the top-down development methodology, testing should occur in the same sequence. However, the hierarchy chart (Figure 2) does not specify the order in which modules should be coded and tested; i.e., all modules on the same level or all modules within a leg of the hierarchy chart. The approach to take is the one which enables you to discover major problems as soon as possible. The development and testing sequence should consider the dependency of modules on data produced by other modules; i.e., execution order dependency.

In this approach, the functional specification of a system and the program code are verified as being correct before going to a lower level of specification. Because modules are developed from the top down, there is continued integration and testing as program modules are developed.

APPLICATION OF SOFTWARE METHODOLOGIES TO SAMT COMMON CORE FUNCTIONS

In order to illustrate our implementation of various software methodologies, an example will be given by addressing one particular module ; namely, the STATE PROCESSOR. The first step in developing the SAMT Common Core software was the definition of functions to be implemented. The short range goals mentioned previously formed the basis of our requirements. These were further refined, resulting in the hierarchy chart shown in Figure 3. Top-down development on the STATE PROCESSOR decomposed it into four modules: CONDITION EVALUATOR, ACTION ROUTINES, SCANNING ROUTINE and ERROR ROUTINE. The development of these modules was not a parallel effort due to the data dependencies among them. Analysis of this dependency and invocation sequence shown in Figure 3 demonstrated that the schedule of development should have the scanning routine first, condition evaluator second, followed by the action and error routines.

The sequence invoked by the STATE PROCESSOR starts with a CALL to the SCANNING routine. The SCANNING routine is responsible for reading the status of all inputs to the SAMT. The inputs to the SAMT are processed via the I/O HANDLER. Figure 3 shows that the dependency of the STATE PROCESSOR on data produced via the I/O HANDLER will dictate the scheduling and criticality of modules for development. The CONDITION EVALUATOR module determines whether the correct sequences of operations have been performed, and if so, causes the ACTION routine to provide the appropriate response. The ACTION module generates the outputs from SAMT and is part of the I/O HANDLER. This interdependency between the OPERATING SYSTEM and MAINTENANCE ALGORITHM is definitized by using the top-down development methodology. Critical modules can now be identified, coded, and tested, so as to insure a smoothly managed software project.

CONCLUSION

The SAMT Common Core Program, briefly discussed in this paper, being pursued at Grumman is in recognition of the growing trend toward the use of simulation for maintenance trainer applications. The computer software approach has a most significant impact on the simulation. As a consequence, careful study and consideration have gone into the selection of designs and methodologies which will yield the most efficient and effective results. This paper has defined the software methodologies to be used in the design and implementation of small maintenance trainers for the near future. A software module approach was developed which satisfies this goal. Further study, testing and refinements of the software design and techniques menus as discussed in this paper, will permit Grumman to expeditiously meet the needs of the maintenance training community with high quality, standardized and cost effective products.

REFERENCES

- "Program Testing," Computer, Volume 11, No. 4, April 1978
- Gildersleeve, Thomas R. "Data Processing Project Management," Van Nostrand Reinhold Company, 1974
- Military Standard Trainer System Software Development," MIL-STD-1644 (TD), 7 March 1979
- Ulrickson, Robert W. "Getting the Bugs Out of Your Software," Electronic Design, 7 June 1977.

ABOUT THE AUTHORS

MR. MICHAEL C. DASHOW is a Senior Software Engineer at Grumman Aerospace Corporation in the Software Systems Department engaged in system specifications algorithm development and hardware/software tradeoff studies. He has prior experience with operations systems design, computer systems analysis including diagnostic and test software, and Atlas compiler developments. He is currently active in the advanced development area of trainer systems. He has a B.E.E. degree from City College of New York and a M.S.E.E. degree from Polytechnic Institute of New York.

MR. MICHAEL A. KOGAN is a Technical Specialist at Grumman Aerospace Corporation in the Software Systems Department engaged in various aspects of computer systems specifications, analyses, and software developments. His past responsibilities have been in technical management, with particular emphasis on computer evaluation, including peripheral equipment, interface design, and microprocessor systems design, development, evaluation, and application. He is currently assigned to the advanced development of trainer systems. He holds the M.E. and M.S.E.E. degree from Stevens Institute of Technology.