ADA STRUCTURAL MODELING DESIGN EXPERIENCE FROM AN ENGINEERING MANAGEMENT PERSPECTIVE

T. Michael Moriarity
AAI Corporation
Baltimore, Maryland

ABSTRACT

Ada Structural Modeling (ASM) is a software development concept that emphasizes the architectural aspects of a real-time software system. The concept was developed by the Aeronautical Systems Center, Wright-Patterson Air Force Base, with assistance from the Software Engineering Institute, Carnegie Mellon University. The concept was originally developed for flight trainers, but has recently been used to design the Simulator for Electronic Combat Training (SECT), a high-fidelity, classroom trainer used to train Air Force Electronic Warfare Officers.

As might be expected, the infusion of a new technological concept presented the development team with numerous technical challenges and opportunities. While the specific technical responses to those demands are of interest to the design analyst, the effect of the responses on the program is of interest to the engineering manager. This paper reviews the ASM design experience on SECT, summarizes its effects on the program, and documents lessons learned for using ASM concepts on future programs.

About the Author

Mr. Michael Moriarity is a Principal Development Engineer in the Training and Test Systems business unit at AAI Corporation. He is currently the project engineer on the Simulator for Electronic Combat Training (SECT) program. Mr. Moriarity holds a Bachelor of Science degree in Mathematics from the University of Minnesota and a Master of Engineering Administration from George Washington University. Previous to his current position he was responsible for software development on the A-6E SWIP Weapon System Trainer, EF-111A Operational Flight Trainer, Navy Electronic Warfare Trainer System, and the defensive instructional subsystem on the B-52 Weapons System Trainer.

ADA STRUCTURAL MODELING DESIGN EXPERIENCE FROM AN ENGINEERING MANAGEMENT PERSPECTIVE

T. Michael Moriarity
AAI Corporation
Baltimore, Maryland

BACKGROUND

Structural Modeling

The structural model concept was developed through the collaboration of software engineers from the Air Force Aeronautical System Center (ASC), the Carnegie Mellon University Software Engineering Institute (SEI), and several training system contractors. The effort was started in 1986 and has developed into a working concept that emphasizes structural aspects of a software system. The concept is documented in a number of informal white papers and was the subject of a tutorial at the 1992 I/ITSEC conference. The concept has been used on several large Air Force trainer programs including the B-2 Weapons Systems Trainer, the C-17 Aircrew Training System, and the Special Operations Forces Aircrew Training System. It is currently being used on the Simulator for Electronic Combat Training (SECT) system. This paper describes the SECT project's design experience with structural modeling.

The development of structural models was motivated by the desire to minimize software maintenance costs while addressing users' training needs in an environment of increasingly complex aircraft and training missions and, also, to respond to users' requests for modifications to the training system. A structural model is defined as an object-based application framework, developed at the design level. Object-based means that the design elements within the application framework, the training system software, are derived using the concepts of objected oriented analysis and design. Developed at the design level means that the design element specifications are partitioned to code units and are defined to a level of detail such that implementation, or coding, can begin. Because trainer systems using structural models to date have used the Ada language, the design level has usually taken the form of compilable Ada specifications and Ada based Program Design Language (PDL). Also, because Ada has been used in all of the structural model implementations the concept has often been called Ada Structural Modeling (ASM).

In its simplest form, ASM defines a small set of structural elements that can be used repeatedly to define the structure of the overall system software. The structural elements are defined for two broad areas or layers: an executive layer and an application layer. The executive layer provides the general coordination services required by trainer system software. This includes the sequencing of periodic processing and the dispatching of aperiodic events. Structural elements provided in the executive layer include sequencing routines, calling order lists, and queue handling logic. The application layer contains the trainer software subsystems that perform the unique trainer simulation tasks such as the simulation of flight controls, engines, avionics, etc. All of the application level subsystems are identical to each other in terms of structure and are made up of structural elements such as subsystem controllers,

subsystem objects, import areas, and export areas.

Another important facet to ASM is the partitioning strategy used to decompose the training system software into its application level subsystems. ASM uses object oriented design techniques to map application level subsystems and their objects to real-world counterparts. For example, an altimeter application level subsystem with its associated pneumatic, electrical, and indicator objects has a direct correspondence to a real-world altimeter subsystem. It is thought that by closely mapping the software subsystems to their real-world counterparts, changes in the real-world systems will result in changes only in the changed object. Also, proper mapping can reduce the complexity of the simulation problem. In the case of the altimeter system with its objects, the loss of electrical power or the activation pneumatic, electrical, or indicator malfunctions can be easily simulated.

Proponents of ASM believe that the repetitive use of a small number of structures improves integratability and maintainability of a software system. Likewise, a partitioning strategy based on real-world objects improves trainer system software functionality and provides a means of coping with the increased complexity of hardware and software technology. ASM, it is believed, through its reduced set of structures and real-world based partitioning also greatly facilitates the communication of design information throughout the system life cycle.

The SECT Program

SECT is a electronic combat mission simulator used to accomplish primary and mission simulator training in the Air Force Air Education and Training Command's (AETC) Electronic Warfare Officer Training Course. Primary training includes threat recognition, analysis, and exploitation using simulated electronic combat equipment. Mission training includes penetration, standoff jamming/direct support, electronic intelligence collection, and suppression of enemy air defense operations. SECT consists of six student stations in which simulated electronic combat equipment panels are displayed on CRTs. The student interacts with the equipment panels via touch screen controls and is able to receive signals, display them on analysis equipment, and counter them through the use of jamming or expendables. SECT also has a two-position Instructor Console and a Training System Support Center. Figure 1 is a layout of the SECT system. The SECT program is managed by ASC, and the

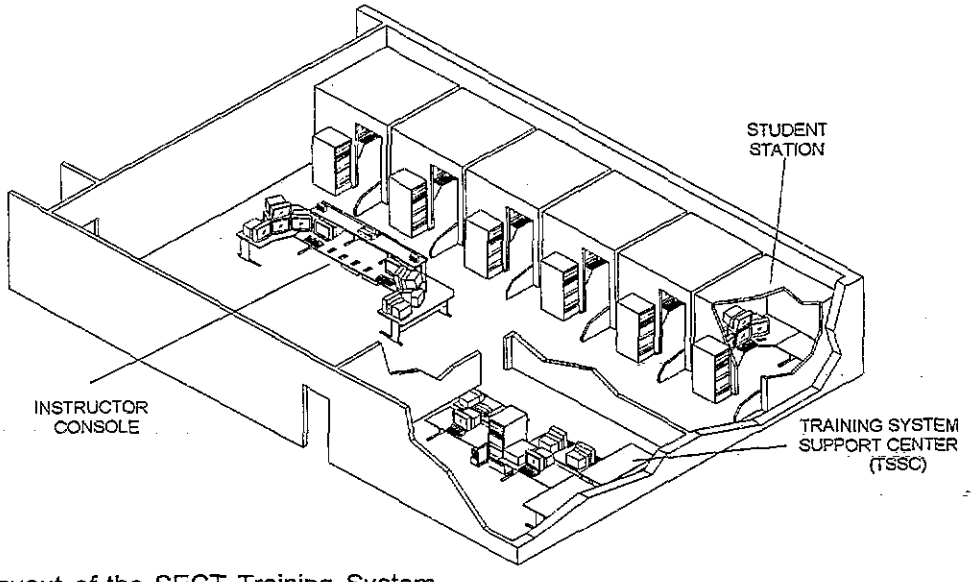

Figure 1. Layout of the SECT Training System

statement of work requires the use of "object oriented methods and Ada Structural Modeling."

Shortly after the award of SECT, an ASM Working Group was established, with representatives from ASC engineering, SEI, and the AAI project team. This working group was the primary means of transferring ASM technology as developed on previous programs to the AAI project team. Sessions early in the SECT program consisted of tutorials presented by ASC and SEI engineers and reviews and discussions of existing white papers. Later sessions consisted of working sessions where specific SECT related design questions were posed, alternatives considered, and design approaches selected. Toward the end of the detailed design phase, as the Critical Design Review (CDR) approached, members of this group participated in subsystem design walk-throughs.

At the time the SECT contract was awarded, ASM techniques had been used on several other large trainer programs. In each of the earlier trainer programs ASM was developed around a flight simulator. SECT represents a departure from these earlier models in that, while SECT includes the concept of a student flight platform or ownship, the student ownship is of secondary importance. The primary entities in SECT are the electronic combat environment and the student's simulated electronic combat equipments. The ownship flight model in SECT is quite simple and basically provides the student with a position and an attitude in the electronic combat gaming area. The aerodynamic handling properties of the simulated ownship are unimportant as long as the ownship's speed, turn rate, and climb/dive rate are similar to the training mission aircraft.

SECT Ada Structural Modeling

The SECT ASM was developed from the earlier flight simulator based ASM efforts. The major elements from the earlier models were retained and relatively minor changes were made for the electronic combat trainer. The concept of the executive layer and application layer is unchanged. The executive layer is made up of an executive and event queue. The executive determines the calling order of the application level subsystems and the rate at which those subsystems are called. The executive, depending on the training system mode (initialize, run, freeze, reset, etc.), calls a predefined list of subsystem controller entries. The event queue provides logic to queue up asynchronous events, or commands, and calls the event processor entry of the appropriate subsystem.

The application layer is made up of a number of subsystems with identical structures. Each subsystem has a subsystem controller which in turn controls the execution of the objects specific to the subsystem. Subsystem control is exercised through two periodic entries (import and update) and/or three aperiodic entries (initialize, reconfigure, and process event). Each of the objects has corresponding entries which are called by its system controller.

Interfaces among subsystems are restricted to two constructs: import/export areas and events. Import/export areas are used largely for the passing of periodically updated data among subsystems. Subsystems that create data place the data in the subsystem's export area. Subsystems that use data have an import element that accesses ("withs" in Ada terminology) the export areas of interest and uses the data needed. Events are used for asynchronous actions and often take the form of a command; are generated by the lesson script, instructor actions, or other subsystems; and

generally pass little data and cause an immediate action to occur.

Figure 2 shows a simplified diagram showing the major components of the SECT ASM using a chaff/flare dispenser as a representative equipment model subsystem.

A major extension to the SECT ASM over previously developed models is the use of two executives running at two different update rates. Flight simulators, as compared to electronic combat trainers, are made up of a large number of subsystems that require a moderate iteration rate, say 20 Hz, and each subsystem requires a low to moderate amount of computational processing. The iteration rates for these subsystems, which typically model flight controls, platform motion, avionic responses, etc., need to be sufficiently high to provide realistic responses to student/pilot control inputs. Often, the computational requirements for each subsystem is low to moderate, because the student/pilot can only provide a limited number of inputs to the system in a given iteration. It is noted, of course, that certain flight trainer subsystems such as visual systems or radar landmass simulators do not necessarily follow these generalities.The characteristics of flight simulators generally lead to a single executive structure where many subsystems are processed within the period of one frame and the iteration rate of a subsystem is determined by the number of frames per second in which the subsystem is executed. For example, in a trainer with a basic iteration of 20 Hz all processing, including required spare processing time, has to be accomplished in a 50 msec interval or frame. Subsystems that require a 20 Hz update rate are executed each frame, those that require a 10 Hz update rate are executed every other frame, and those that require a 1 Hz update rate are executed once every 20 frames.

A difficulty with this process is that subsystems that require a 1 Hz update rate but cannot be allocated sufficient time within a given frame need to be time sliced; that is, the task must be subdivided to run over several frames but the subsystem as a whole is executed only once per second. The normal flight simulator often has a large number of subsystems that have to run at the higher iteration rates and time slicing is often minimized. This allows flight simulators to use single rate executives rather efficiently and all processing is performed at the basic iteration rate or some evenly divisible integer divisor of the iteration rate, such as 10, 5, 4, 2 or 1 Hz in the case of a 20 Hz executive.

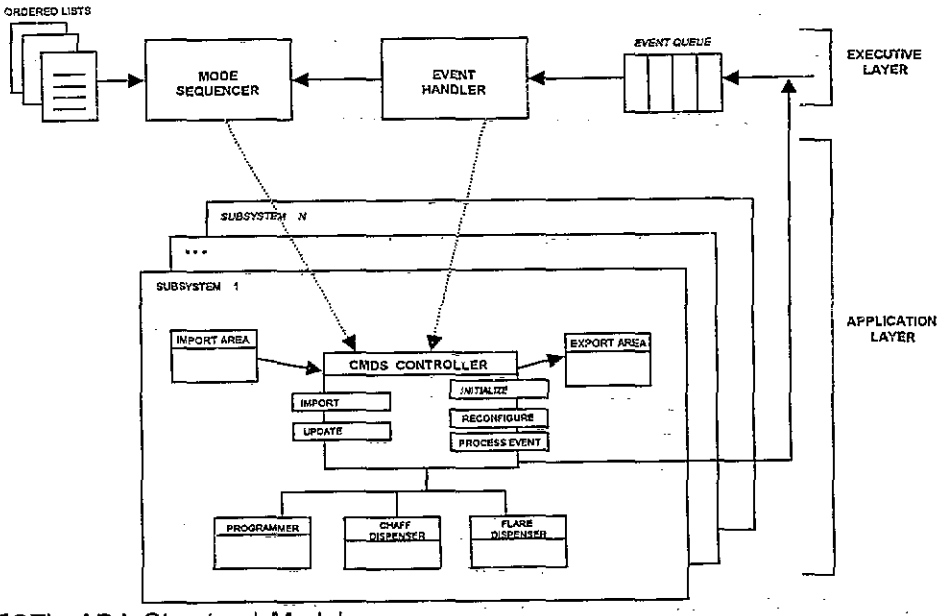

Figure 2 - SECT's ADA Structural Model

The requirements of an electronic combat trainer are somewhat different from those of a flight simulator. The simulation of electronic combat equipment is similar to a flight trainer's simulation of avionic subsystems. The electronic combat equipment subsystems are executed at moderate iteration rates and often have low to moderate computational requirements. These subsystems, like the flight simulator, must respond in a realistic manner to student inputs. The electronic combat environment (ECE), however, is a different matter. Typically, the changes in the ECE occur relatively slowly. New signals coming into view, mode changes on a radar system, and range effects due to changing distance between transmitters and receivers all happen relatively slowly. Updates of 1 Hz or less are usually sufficient for these types of simulations. After a change does occur, however, a great deal of computational processing is required. This type of simulation is best supported by an executive running at a slow update rate such as 1 Hz. At 1 Hz the update rate more closely matches the real-world requirements and the 1000 msec interval allows a greater portion of the processing to be completed so that time slicing does not have to be performed.

The dual iteration requirement resulted in SECT providing a 1 HZ and a 20 Hz executive. This decision added complexity to the executive layer in that additional logic had to be designed to allow for event queues to be processed by both executives. Also, coordination logic had be provided to ensure data integrity in instances that data created by a subsystem called by one executive was valid when used by a subsystem called by the other executive.

SECT'S EXPERIENCE WITH ASM

ASM Learning Curve

The time to learn ASM concepts and use them to develop the SECT's detail designs took longer than estimated. The AAI project team originally based its estimates on a two-step design process: first, acquire a working knowledge of the ASM concepts and techniques, and, second, use the techniques to design the training system software. It was thought that after familiarity with the new techniques was achieved, the remaining design activity was simply to apply the new techniques to a familiar problem. The effort, it was thought, would be similar to replacing a functional design process with an object oriented design process.

The flaw in this thinking was that the ASM design process required at least three steps: first, acquire a working knowledge of the ASM concepts and techniques; second, use the techniques to design the ASM structural elements, and, third, design the trainer software system using only those defined structural elements. The key issue impacting schedule was that two largely sequential design activities were required when applying ASM to a new software domain. The structural elements needed to be designed before the software system could be designed.

Structural Design Validation

One method to validate the structural design is to actually design a portion of the system that encompasses each of the unique aspects of the system. Furthermore, to validate the structural elements, it is important to prototype key portions of the system. This should be done early in the detail design phase so that the structural model can be stabilized before the majority of the detail design is started.

The lesson learned by the SECT project was two-fold. One, get prototyping done early; and two, carefully choose the portion of the system to be implemented for validation purposes. By completing the prototyping early, there will be minimal design effort complete; and, therefore,

minimal design changes resulting from structural model changes. Carefully choosing the validation cases can reduce effort and pay high benefits. It is important to pick cases that validate the key system wide control concepts rather than subsystem unique problems.

System Visibility

ASM provided excellent visibility of the trainer system software structure after the structural elements were defined. Because the software system was built up from replicated structural elements, it was easy to get a grasp on the overall system structure. The increased visibility was evident when new analysts were assigned to the project or when existing project personnel were assigned new tasks. In the first instance, new programmers, after they understood the nature of the structural elements, developed a good working understanding of how the whole system worked from a subsystem control and interface standpoint. The benefit was even greater when existing project personnel were assigned new systems; they already understood the structure of their new subsystems.

It should be noted, however, that just because an analyst understands the structure of the system, it doesn't necessarily mean they understand the functionality of system. It is one thing to understand how the chaff/flare subsystem is called and how communication is performed, but another thing to realize that the subsystem exists, to know what data it provides or the algorithms it uses.

System Issues Addressed Early

ASM tends to surface high level system control and interfacing problems early. This, of course, is desirable in that it allows correction of these problems early and minimizes the effect on the overall system. The SECT project team uncovered several system level problems early, including a potential data integrity problem that was corrected through a change to the structural model. Had this problem surfaced during integration, it probably would have been difficult to track down and very difficult to correct. A modification to all subsystems would probably have been required. Further, it is possible that, at that late point in a development cycle with its attendant schedule pressures, a compromised correction could have resulted in a less than optimal change that would have impacted system modifiability and maintainability in the future.

The flip side of identifying problems early is that the design phase takes longer and the software staff, to say nothing of their managers, can feel somewhat demoralized in that the design seems never to get done--we keep finding problems.

Software Reuse

Software reuse from previous trainer systems was much less than originally planned. The SECT system was similar to other trainers AAI had built in the past and, while it was realized that the software from these other trainers would have to be rewritten to meet Ada and ASM constraints, it was thought that significant benefits could be garnered from the earlier developed software. The SECT experience has been that while some specific algorithms developed on other trainers were useful as resource data, no software could be reused on SECT from other trainers. The difference between the earlier functionally developed software and ASM requirements was so great that it was determined that the effort to convert old software was greater than the effort to design new software.

In contrast, reusing software developed for one part of SECT on another part of the trainer appears to be easier under ASM than on previous systems. There were several subsystems or parts of subsystems that were developed for use in the student station that

could be used in the Instructor Console or in the Trainer System Support Center. Because the calling sequences and the communication methods were tightly controlled under ASM, it was possible to use the same subsystem to perform similar functions in different trainer stations.

Emphasis on Interface Structure, Not Interface Data

The intrinsic emphasis of ASM is to focus on the structure of the software system. A potential negative aspect of this focus is that other important design issues may not get the emphasis they require. One area where this is prone to happen is in the data structures in the import/export areas. ASM concepts put emphasis in identifying the structural elements by which data is transferred and the coordination of those transfers. Likewise object oriented design, to which ASM is related, emphasizes data hiding. The emphasis on structural aspects of data transfer and on subsystem independence through information hiding can result in the lack of proper attention to the definition of the actual data and its structure that is to be passed from subsystem to subsystem. In SECT this happened to some degree in that design walk-throughs paid considerable attention to how data was passed across interfaces and the kind of data that had to be passed, but not to the structure of the data. As a result, additional definition had to be accomplished prior to the start of coding.

Prototyping

The need to define structural elements early and then validate these structures by using them in portions of the system design resulted in prototyping key aspects of the trainer system software early in the design effort. The early prototyping was very beneficial from several aspects. First and most important, prototyping uncovered flaws in the original designs that were able to be corrected early in the development cycle. In many instances the flaw was corrected before it impacted other designs; in some instances designs were already under way and changes had to be made. These flaws were uncovered in the design of structural elements which in turn affected all subsystems in design. In other cases the changes uncovered flaws in the way the structural elements were used in a specific subsystem. In these cases a body of knowledge was built up that other subsystem designers could use in their designs.

Prototyping was also invaluable in that it gave the project team experience in using the software development system early in the program. Development system tools, responses, and resources were exercised and monitored in a realistic manner. Configuration management, archiving, and various housekeeping processes were also verified.

The prototyping effort evolved into a basic test harness for unit testing and later into an integration harness. The basic prototype system consisted of most of the executive layer and several of the simpler subsystems in the application layer. The prototype system supported two different rate executives, interprocessor memory management, event queue logic, and all trainer modes. The application layer had all subsystems stubbed out and several subsystems completed. The prototype system was demonstrated to the customer at various times throughout the development cycle. The most extensive demonstration was at CDR, which strengthened the emphasis on the user aspects of the system

Preliminary Design Review Effects

The emphasis during Preliminary Design Review (PDR) and Critical Design Review (CDR) changed from traditional non-ASM projects. During the SECT PDR the primary focus was the design of the structural model and its

minimal design changes resulting from structural model changes. Carefully choosing the validation cases can reduce effort and pay high benefits. It is important to pick cases that validate the key system wide control concepts rather than subsystem unique problems.

System Visibility

ASM provided excellent visibility of the trainer system software structure after the structural elements were defined. Because the software system was built up from replicated structural elements, it was easy to get a grasp on the overall system structure. The increased visibility was evident when new analysts were assigned to the project or when existing project personnel were assigned new tasks. In the first instance, new programmers, after they understood the nature of the structural elements, developed a good working understanding of how the whole system worked from a subsystem control and interface standpoint. The benefit was even greater when existing project personnel were assigned new systems; they already understood the structure of their new subsystems.

It should be noted, however, that just because an analyst understands the structure of the system, it doesn't necessarily mean they understand the functionality of system. It is one thing to understand how the chaff/flare subsystem is called and how communication is performed, but another thing to realize that the subsystem exists, to know what data it provides or the algorithms it uses.

System Issues Addressed Early

ASM tends to surface high level system control and interfacing problems early. This, of course, is desirable in that it allows correction of these problems early and minimizes the effect on the overall system. The SECT project team uncovered several system level problems early, including a potential data integrity problem that was corrected through a change to the structural model. Had this problem surfaced during integration, it probably would have been difficult to track down and very difficult to correct. A modification to all subsystems would probably have been required. Further, it is possible that, at that late point in a development cycle with its attendant schedule pressures, a compromised correction could have resulted in a less than optimal change that would have impacted system modifiability and maintainability in the future.

The flip side of identifying problems early is that the design phase takes longer and the software staff, to say nothing of their managers, can feel somewhat demoralized in that the design seems never to get done--we keep finding problems.

Software Reuse

Software reuse from previous trainer systems was much less than originally planned. The SECT system was similar to other trainers AAI had built in the past and, while it was realized that the software from these other trainers would have to be rewritten to meet Ada and ASM constraints, it was thought that significant benefits could be garnered from the earlier developed software. The SECT experience has been that while some specific algorithms developed on other trainers were useful as resource data, no software could be reused on SECT from other trainers. The difference between the earlier functionally developed software and ASM requirements was so great that it was determined that the effort to convert old software was greater than the effort to design new software.

In contrast, reusing software developed for one part of SECT on another part of the trainer appears to be easier under ASM than on previous systems. There were several subsystems or parts of subsystems that were developed for use in the student station that

structural elements. An important, secondary focus was the partitioning of the trainer system software into its subsystems. A third focus was the identification and development of modeling algorithms to be used to accomplish the simulation. The priority of these three tasks was dictated by the needs of the ASM design process itself--the structure needs to be defined before the subsystems can be defined, and the subsystems need to be defined before the algorithms can be identified.

This had two effects at PDR. The first is that the relative emphasis of the design disclosure met the expectations and needs of the ASC software engineering staff, but did not provide the user with the design information they were looking for. The user's interest was almost the opposite of the engineering staff's. The user would have liked to reverse the priority order--algorithms, subsystem partitioning, and structure.

The second effect was that flaws in the application of structural model concepts to the trainer system software design affected the partitioning of the trainer system, which in turn affected the selection of subsystem algorithms. These issues were worked during the detailed design phase.

In retrospect, a two-step preliminary design review would have worked better. The first review would have reviewed the structural model and its elements. This review would have been of primary interest to the ASC software engineers. A second review would have covered the partitioning and algorithmic aspects of the trainer system software. This review would have been of equal interest to the engineering and user communities.

Critical Design Review Effects

The emphasis during CDR was on trainer system design aspects from the user's point of view. Two circumstances encouraged this approach. The first was simply to provide the user with design data that was needed to ascertain that the system met all training needs. The user emphasis at CDR helped compensate for the information not attained at PDR. The second, and more important aspect, was that the ASC engineering staff already had a good understanding of the trainer system software design before CDR. Between PDR and CDR the SECT system structural model was documented in an "Ada Structural Model Report", all subsystems were designed in strict adherence to the documented model, and the Air Force software engineers were invited to sit in on the internal design walk-throughs. This active participation by Air Force engineers enabled Air Force concerns to be identified early and addressed by the SECT design team in a timely manner. This process also gave the Air Force engineering staff a very high degree of visibility into the SECT design as it was unfolding. By the time the formal CDR was held, the detailed aspects of the software design had already been disclosed to those who had the greatest interest in that design. The resulting CDR then had a greater appeal to a wider range of participants.

SUMMARY

SECT's experience with ASM during the project's design phase raises some concerns but confirms that the goals of ASM are achievable. The concerns raised are the increased time to accomplish system design and the potential difficulty in reusing previously developed non-ASM software. The increase in the design schedule experienced by SECT was related to the need to learn a new design process, to extend the structural model concept into a new class of trainers, and to validate the extended structures. It is anticipated that any design team unfamiliar with ASM concepts and applying the concepts to a new class of trainers

would experience similar results. A team, having once gone through the process, and using a previously developed structural model, should experience design times equal to, or better than, those achieved using more traditional design techniques.

The inability to capitalize on previous software designs was disappointing, but in view of the current state of software reusability, perhaps not unexpected. The formalized structure imposed by ASM does reduce the probability that previous non-ASM designs will be usable in ASM projects. The good news is that based on reusability within SECT, there is greater hope for reusability among ASM projects that use common structures.

SECT's experience shows that ASM does improve system visibility and that the increased visibility enables system level design flaws to be identified early in the design cycle. The increased visibility improves communications among designers and between designers and reviewers. The fact that ASM strongly encourages prototyping increases a deeper understanding of the evolving system. It is anticipated that the higher level of visibility will make future maintenance and modification activities more efficient in the ASM based system.

The increased design time and increased visibility into the system software structure does suggest that the content and timing of formal reviews such as PDR and CDR should be revisited. For projects that are developing new ASM structures or are applying ASM to a new class of trainers, an early, formal review of the structural model is recommended. This review should be followed by a more traditional PDR that looks at partitioning and algorithmic designs. Projects that have an active, ongoing ASM working group throughout the design phase should consider shifting the emphasis from detailed software designs at formal CDR to broader more functional design disclosure.