SOFTWARE MANAGEMENT OF A COMPLEX WEAPON SYSTEM SIMULATOR

ALBERT S. GOLDSTEIN AND WENDELL J. NEWELL
The Boeing Company
Wichita Division

INTRODUCTION

This paper presents the history and lessons learned in the development and implementation of the computer programs for a large complex Weapon System Trainer (WST). The WST is a high fidelity simulator of the B-52 and KC-135 crew stations (Figure 1). The WST includes visual, motion, sound, and a highly flexible instructional system. The contract for the development of a production prototype unit began in mid-1977. The development team consisted of the Boeing Company, Wichita as integrator and several subcontractors responsible for the various stations and systems as indicated in Figure 2.

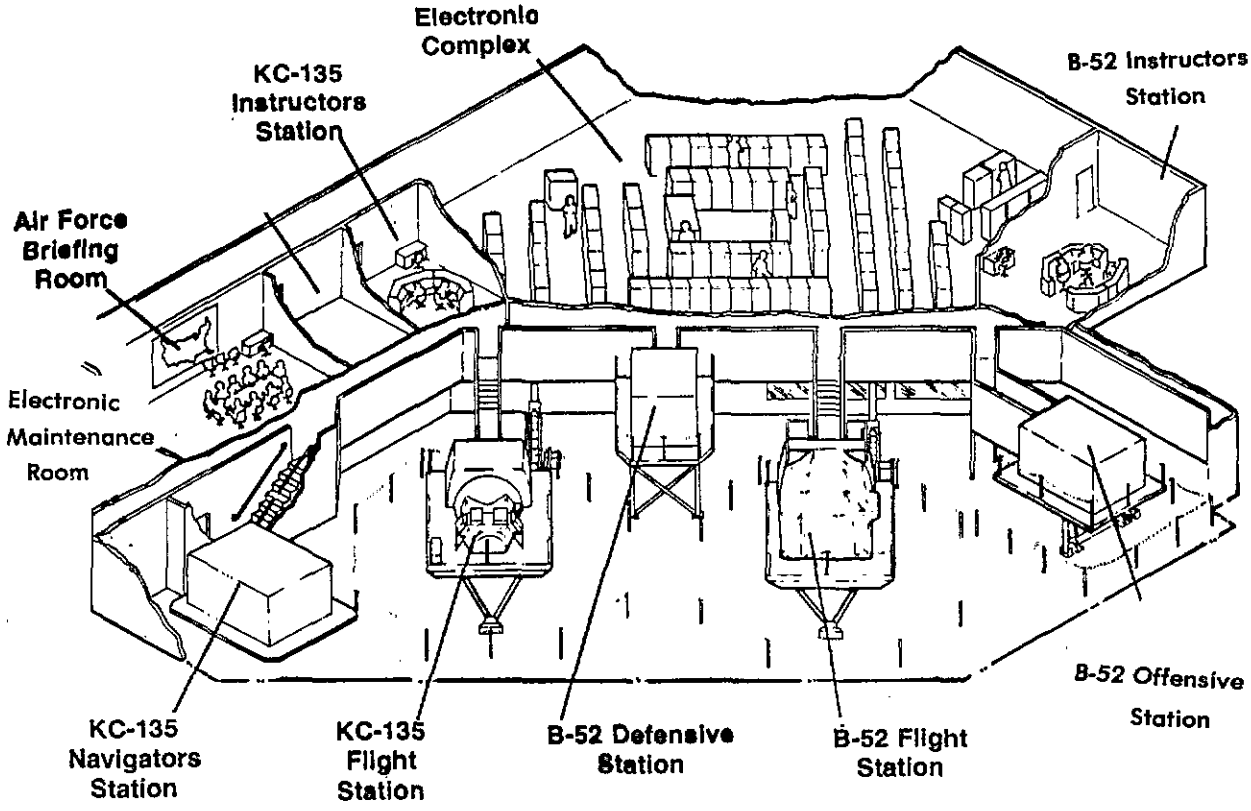


FIGURE 1 B-52/KC-135 WST TEST COMPLEX

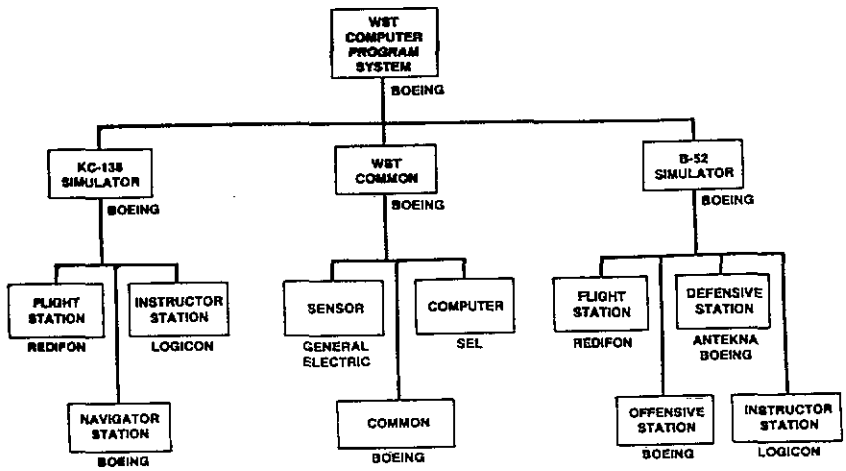


FIGURE 2 WST COMPUTATIONAL SYSTEM

COMPUTER PROGRAM SYSTEM

The WST computer program system is composed of over 600 Real-Time Operating Programs (RTOPS) and support software. Standards were established to ensure modularity so that most RTOPS are between 100 and 400 lines of codes. For each 4 to 5 lines of code, there are 3 lines of comments. In addition, there are several large programs of one to two thousand lines of codes. When support programs are included, the total lines of code exceed one million. A structured Fortran preprocessor was selected and assembly language programming was allowed only in rare special cases. This S-Fortran processor allowed direct implementation of structured design and was easy to read and maintain; it increased programming efficiency and increased program reliability.

The applications software can be characterized as having the elements of a large real-time command and control system (with significant man/machine interfaces) along with the real-time response required of high fidelity simulators. The computer hardware consists of Systems 32/55 minicomputers and associated peripherals. The WST crew stations are designed to operate in independent, mixed and integrated modes (Figure 3). A key interface design for both the simulator operations and software development was the implementation of a shared memory datapool arrangement. This shared memory scheme (Figure 4) makes it possible to change the software design and revise simulator operations with minimum design impact. This datapool design also made it possible to design, code, and test the software so that programmers need use only mnemonic names for all variables without knowing the locations and detailed characteristics of each variable.

MANAGEMENT CONTROLS AND TECHNIQUES

It was recognized at the beginning of the WST Program that the complexity and amount of code, and the division of design responsibilities among team members, required establishment of software standards, measurements, and tracking techniques. Both budget and schedule limitations were severe. Although the team assembled represented some of the most qualified and experienced in terms of design, data, and operations, the software design teams were characterized by young and inexperienced personnel. (Now a seasoned software development group.) This turned out to have a significant advantage, in that the proper management standards, procedures, and tracking techniques could be applied with a minimum of the typical software design syndromes encountered in large complex computer program development projects. Figure 5 shows the similarities between hardware and software design processes used. We were able to resist pressures to shortcut the software development process from both designers and higher management through the reporting and tracking mechanism. We avoided the 90% complete, let me program and forget about design documentation, I'll worry about interfaces later, don't constrain me with programming standards, and that's the way we did it ten years ago syndromes. Computer program technical integrity was maintained through technical walkthroughs on a module basis, frequent design reviews, module tests and verification procedures, subintegration test procedures and documented test results. The documentation mechanism used was an adaptation of the milestone system (Figure 6).

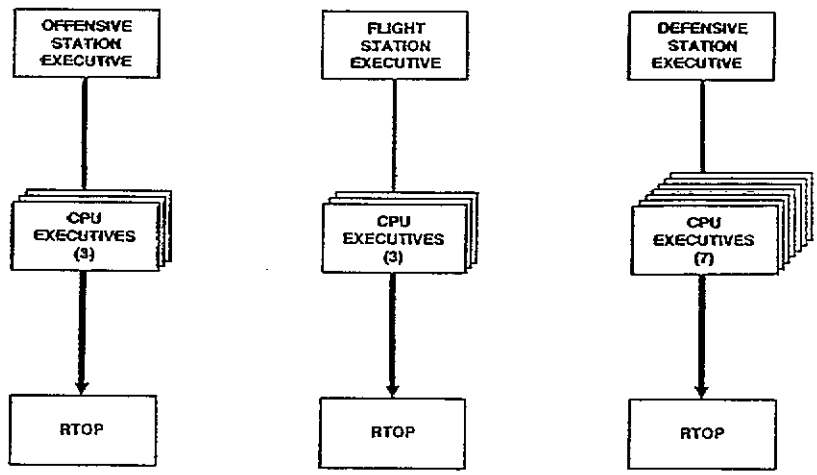


FIGURE 3a B-52 INDEPENDENT MODE

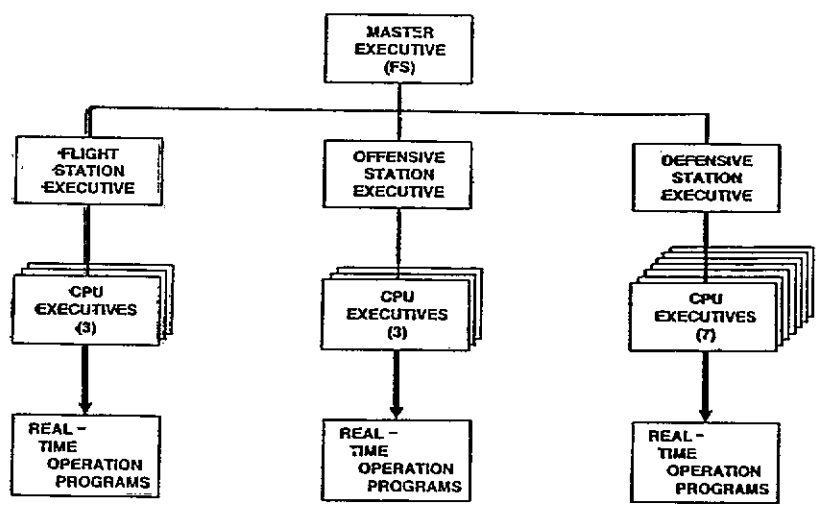


FIGURE 3b B-52 INTEGRATED OPERATION MODE

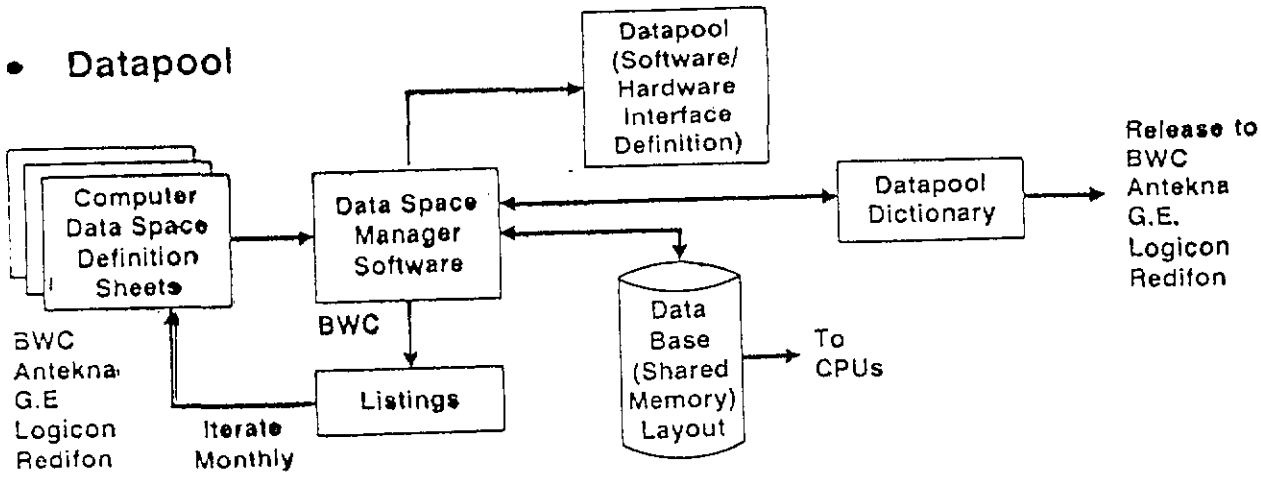


FIGURE 4 SOFTWARE INTERFACE CONTROL

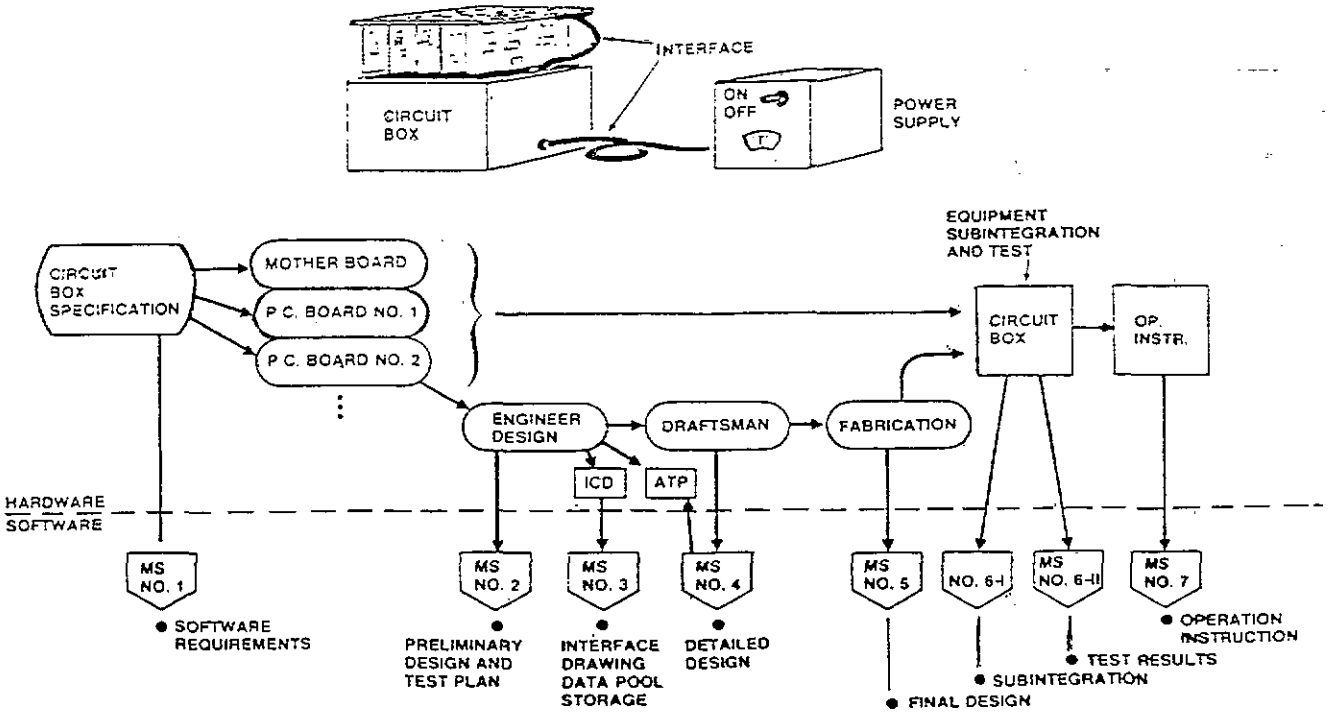


FIGURE 5 SOFTWARE VS HARDWARE

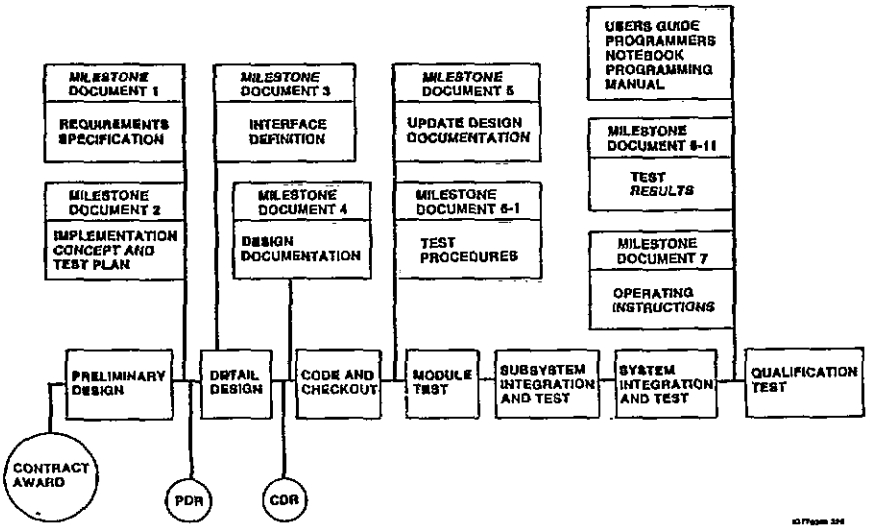


FIGURE 6 MILESTONE DOCUMENTATION

The other key management action was to apply the same engineering design disciplines used successfully by Boeing for hardware and airplane systems integration to the formal software development process shown in Figure 7 (and we stuck to it). We modified the waterfall theory as shown in Figure 8, by providing feedback through the mechanism of completely revised and approved documentation prior to allowing coding, and the implementation of formalized computer program change procedures that provided traceability and control of all changes resulting from module and system testing (Figure 9). A committment was made to include a quality assurance role early in the software development process prior to module testing instead of at the verification test procedure effort late in the program. This helped maintain configuration control and audit trails.

INTERFACE CONTROL

A key mechanism for ensuring that the software design was tied to the simulator hardware design was to use the datapool mnemonics as the hardware and software interface identifier (Figure 10). All wiring connectors and signals were identified and correlated to the variable mnemonics. The datapool was maintained and controlled by the adaptation of a commercially available data base manager program. By insisting that all software development standards, and all interfaces with shared memory be defined among all team members, the tasks of hardware/software integration and test were made possible with a minimum of the problems encountered in this phase of software development.

- **Conduct a formal software development process**

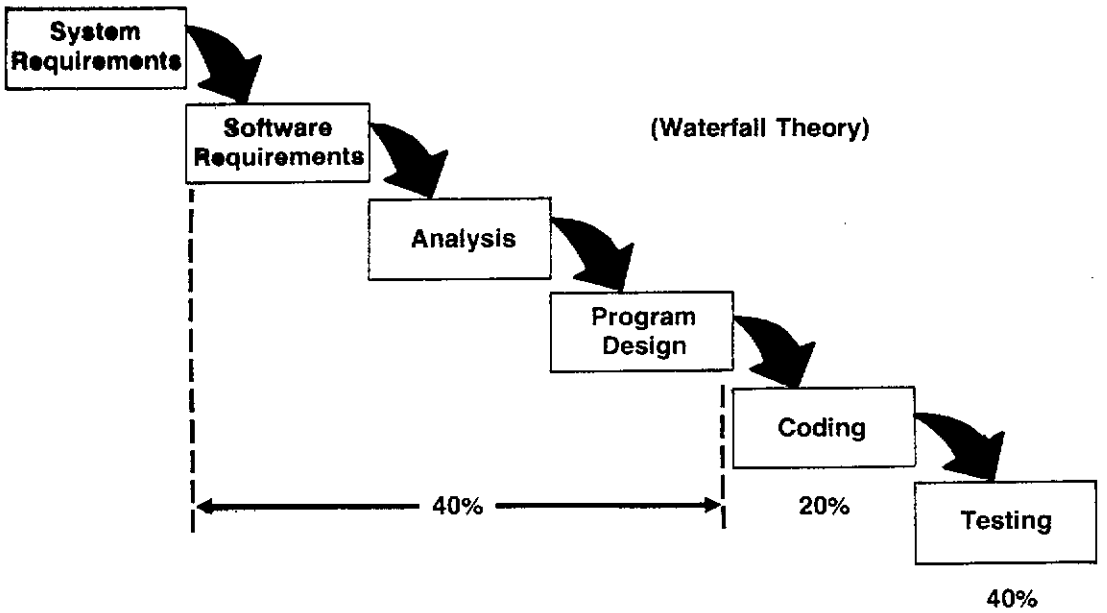
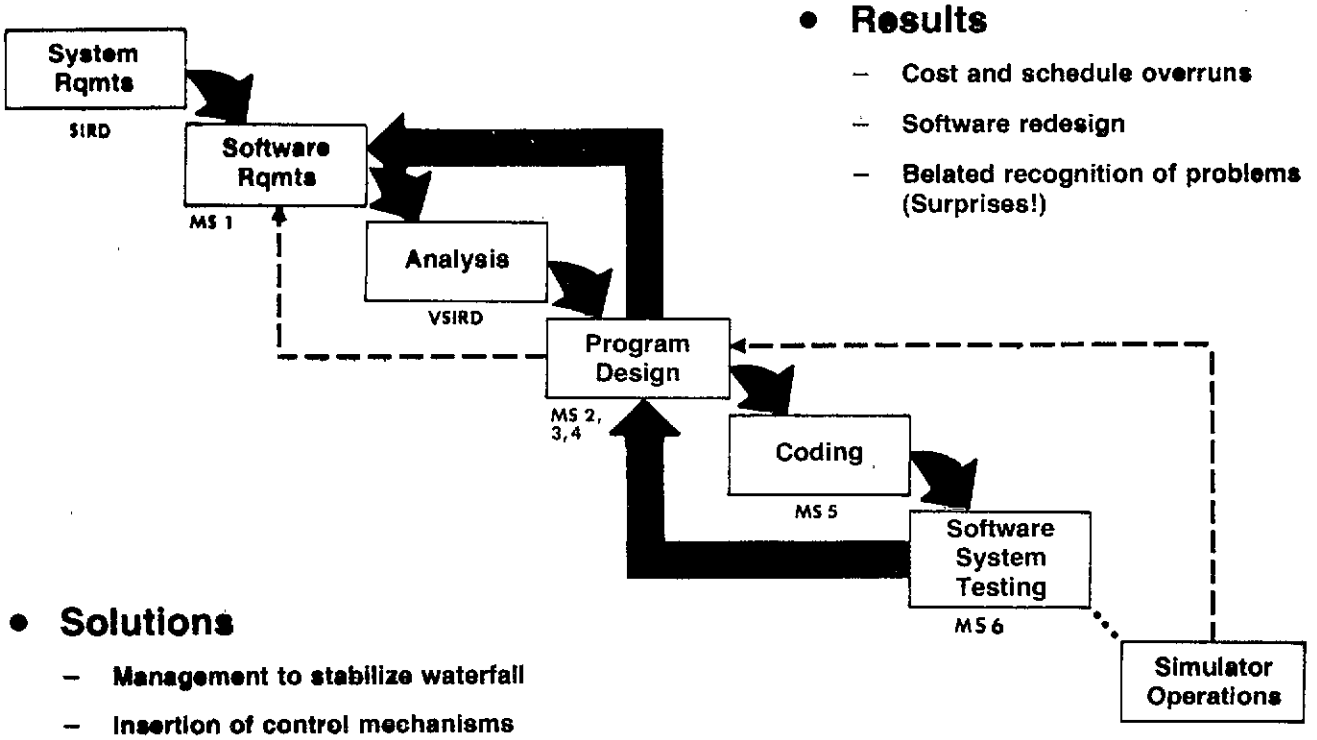


FIGURE 7 FIRST PREREQUISITE TO INTEGRATING SOFTWARE

- **Results**
  - Cost and schedule overruns
  - Software redesign
  - Belated recognition of problems (Surprises!)



- **Solutions**
  - Management to stabilize waterfall
  - Insertion of control mechanisms
  - Engineering discipline applied to software

FIGURE 8 WATER FLOWS UPHILL IN THE SOFTWARE DEVELOPMENT PROCESS

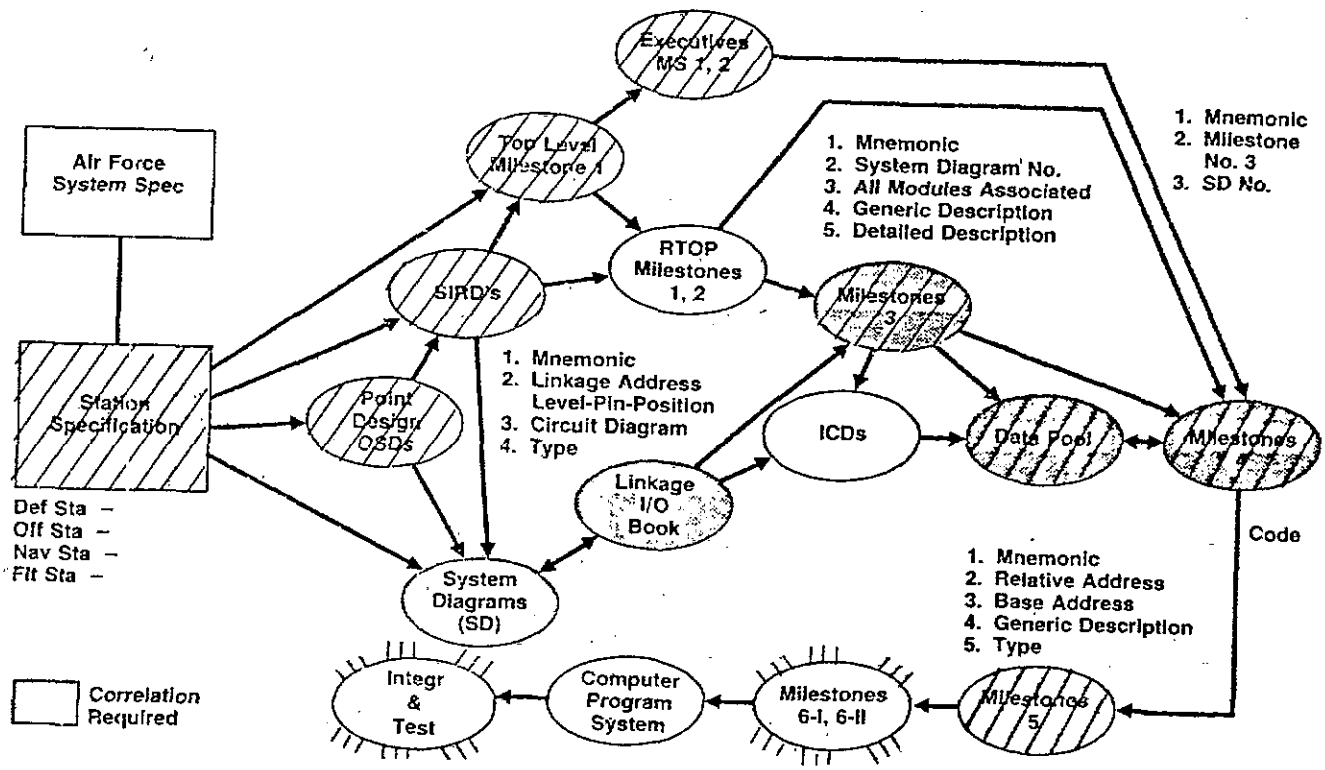


FIGURE 9 DESIGN TRACEABILITY AND DATA CORRELATION REQUIREMENTS

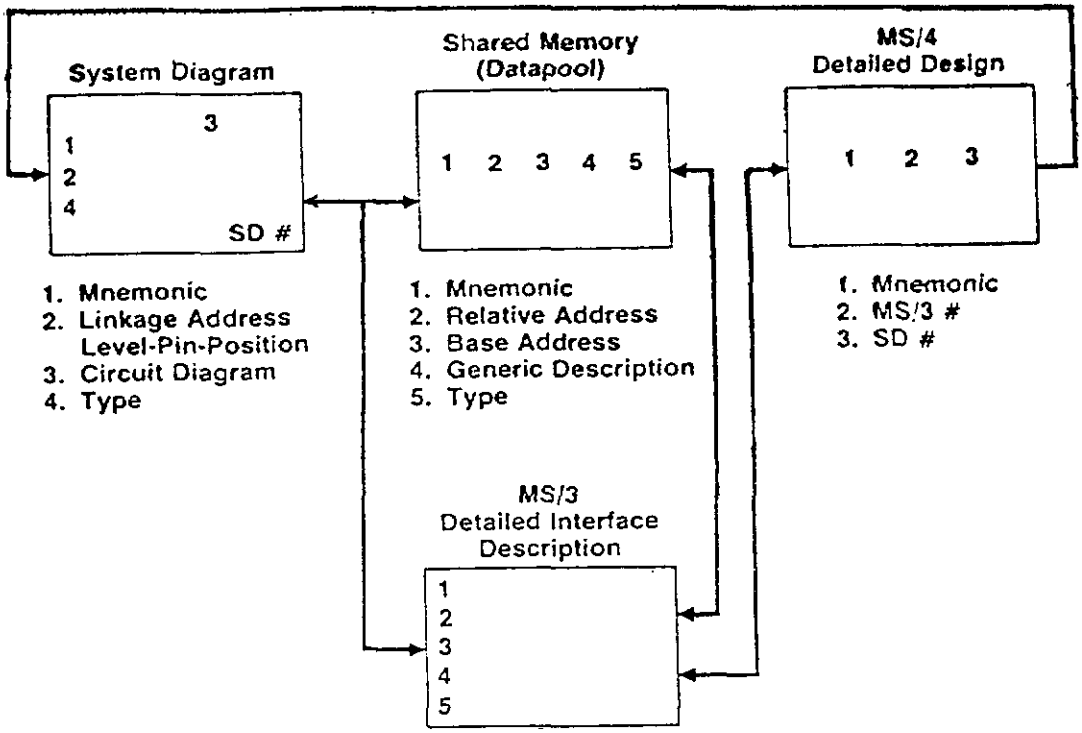


FIGURE 10 HARDWARE/SOFTWARE INTERFACE CONTROL

PROBLEM AREAS

Management Organization

The original organization of the computer program development group consisted of a design organization and requirements and verification organization. There was no single authority for the computational design. Early in the development program, it was recognized that this organization was unworkable. Those portions of the requirements and verification group that were necessary to provide the checks and balances were retained and placed under the computer program design manager. The main function of this group was to ensure the integrity of the design by thorough documentation review and design test. Later in the development schedule, this group shed its role as watchdog and performed most of the module coding and testing functions which proved the modularity concept of the software design. Most important to us, this proved that with the proper design documentation and adherence to programming standards a person other than the designer could code the RTOP, and a third person could test the RTOP. These activities had three effects. First it demonstrated to management that the software was properly documented, maintainable and could be easily taken over by the Air Force. Second, this approach resulted in considerable cost savings as our records indicate. The usual 20% of project funds allocated to coding was reduced to less than 15%, which included the costs expended for module testing. Third, the extent of documentation, modularity, and testing allowed a decision to be made to omit the usual extensive sub-integration of software. This function was combined with the Hardware/Software Integration (HSI) tasks. The result was considerable savings in dollars and achieved a very challenging schedule by allowing overlap of the design, code and test phases of the software development and allocation of personnel to these functions cost-effectively.

Another early organizational mistake was

placing of the documentation of the mathematical models in a systems engineering organization. This organization produced the Software Implementation Requirement Documents (SIRDS) which, although adequately detailed on a module level, failed to include sufficient system integration considerations and test requirements in the math models. This deficiency was overcome by the adherence of the software designers to module design and testing integrity. The lesson learned is that development of the math models should be placed under the software development manager to ensure sufficient interchange with software designers, and maintaining of standards to facilitate subsystem and HSI testing downstream. The Systems Engineering function is to define requirements and determine allocation between hardware and software.

Subcontract Integration

A basic integration problem was the lack of adequate interface definition which resulted in false starts and/or late starts on the software design. Interface control documents including datapool definitions were not properly prepared early enough in the project. Correcting this situation required costly establishment of task forces and development of data base generation and control techniques. Earlier and proper data base management and coordination with team members, all of whom were at least 2000 miles away (including England), would have prevented considerable difficulity in the HSI and test activities. The fact that the basic scheme of shared memory design was sound allowed successful attainment of the software design and integration. Other key techniques for control and integration of subcontractor software was to contractually impose the same standards on them, to provide executive and support software, and to provide configuration control and object verification procedures to support software testing (Figure 11).

Measurement and Tracking of Software Development

As stated above, the early establishment of software design and development standards combined with the modular design allowed tracking of progress at the individual RTOP level. The milestone documentation system shown previously in (Figure 9) also allowed the further tracking of each RTOP at each milestone level. Insistence was maintained on complete and thorough documentation prior to code and test and complete review as shown in Figure 12 (requiring 3 signatures) at the working level. Work charts (Figure 13) were established and updated weekly for each simulator station and used by management to monitor progress and reveal problem areas. As work fell behind, management actions, decisions, and recovery plan were implemented. Extensive tracking by all levels of management on daily and weekly intervals was included and a computerized program planning and control system was established showing the start and completion dates of each event, down to the individual milestone document.

CONCLUSIONS BY BWC

A complex simulator software development project can be successfully accomplished on schedule and within budget by adhering to engineering design disciplines and management controls. There are no shortcuts to design and development of software. The key to design integrity and configuration control is to maintain formal quality assurance through established mechanism as shown in Figure 14. Documentation of design prior to coding is essential. A popular belief that programming is an uncontrollable intellectual exercise, that defies application of engineering disciplines and management control techniques, is erroneous as we have proven in the B-52/KC-135 Weapons Systems Trainer. Detail tracking of software development progress is required using realistic measures of progress such as documentation, code and testing of the smallest definable module.

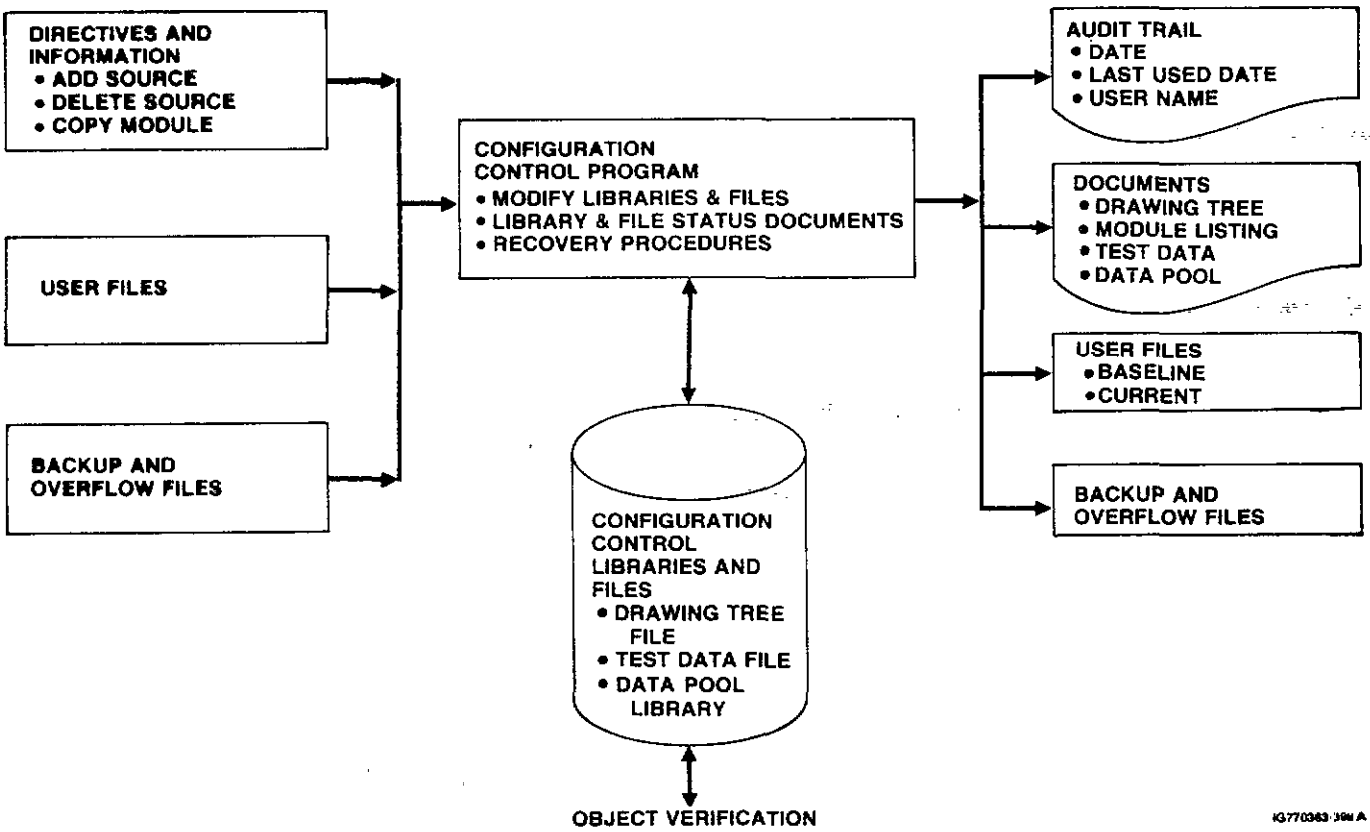


FIGURE 11 CONFIGURATION CONTROL

TECHNICAL WALKTHROUGH REPORT

MODULE NAME _____ DRAWING NO. _____ DATE _____

PRELIMINARY DESIGN _____ DETAILED DESIGN _____ CODE _____

- o Walk through material and work off old comments list.
- o Create comments list.

CHECK LIST

_____ Functional Description
_____ Requirements Referenced
_____ Malfunctions
_____ I/O Definition
_____ Time & Memory Allocations
_____ User Information
_____ Variable Description
_____ Design Meets Requirements
_____ Structure Chart
_____ Design Meets Standards
_____ Flow Diagrams
_____ Test Plan/Procedures
_____ No Extra Requirements
_____ No Division by Zero

_____ Header
_____ Program Name
_____ User Information Consistent
_____ Variable Descriptions
_____ Time & Memory Allocations
_____ Code Represents Design
_____ Commenting
_____ Code Meets Standards
_____ Checkout Sufficient

Decision:

_____ Accept as is
_____ Revise (no further walkthrough)
_____ Revise and schedule another walkthrough

Signatures:

_____ Lead

_____ Presentor (Design Staff)

_____ Reviewer (Tech Staff)

FIGURE 12 TECHNICAL WALKTHROUGH REPORT

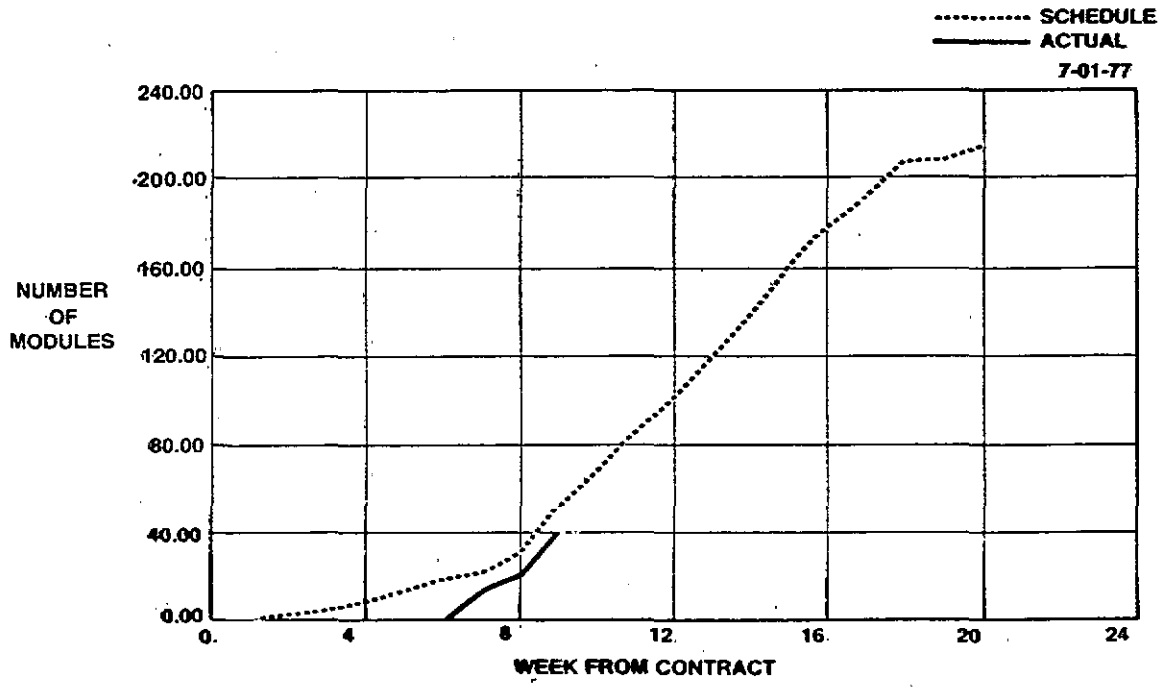


FIGURE 13a TYPICAL MILESTONE 1 STATUS VISIBILITY

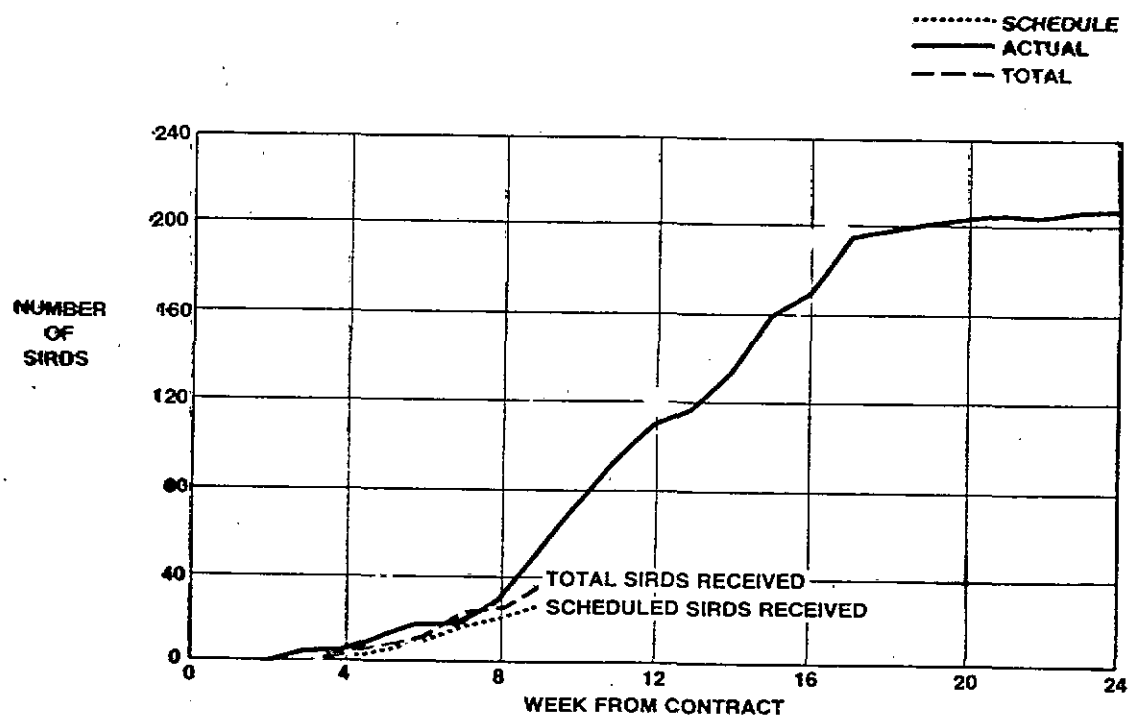


FIGURE 13b TYPICAL SIRD STATUS VISIBILITY

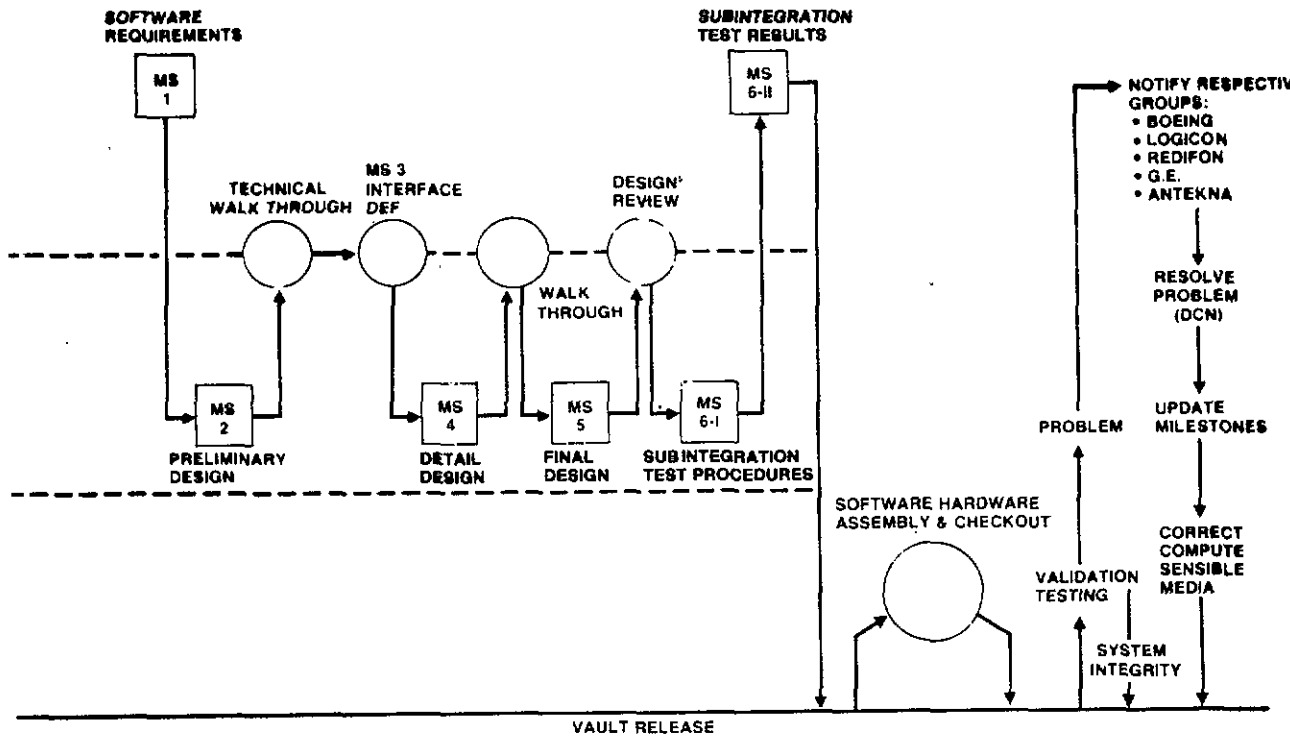


FIGURE 14 COMPUTER PROGRAM QUALITY ASSURANCE

ABOUT THE AUTHORS

MR. ALBERT S. GOLDSTEIN is currently Manager of Design Engineering for the B-52/KC-135 Weapon System Trainer Production Prototype Unit. He has been associated with command control and computer systems for over 20 years in both military and space programs. He has a B.S.M.E. degree from Pratt Institute, M.S.A.E. degree from Syracuse University, and an M.S. degree in operations research from USC.

MR. WENDELL J. NEWELL is currently Chief Engineer of the B-52/KC-135 Weapon System Trainer Production Prototype Unit. He has been previously associated with the B-52, B-1, Airborne Warning and Control System, and Navy Aerospace Ground Equipment Programs in the areas of electronics/electrical design and airborne data systems. He has a B.S. degree in electrical engineering from Kansas State University.