END
DATE
FILMED
5 80
DTIC

1.0

1.1

1.25 1.4 1.6

2.8 2.5

3.2 2.2

3.6

4.0 2.0

1.8

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ASD-TR-78-48

SOFTWARE CONFIGURATION MANAGEMENT
One of the Software Acquisition Engineering Guidebook Series

DIRECTORATE OF EQUIPMENT ENGINEERING
DEPUTY FOR ENGINEERING

JANUARY 1979

DTIC
ELECTE
S APR 1 7 1980 D

B

TECHNICAL REPORT ASD-TR-78-48
Final Report

Approved for public release; distribution unlimited.

AERONAUTICAL SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

80   4 17 081

RICHARD W. ITTELSON,
Technical Advisor
Directorate of Equipment Engineering

RICHARD J. SYLVESTER,
ASD Weapon Systems/Computer Resource
 Focal Point
Deputy for Engineering

FOR THE COMMANDER

JOHN S. KUBIN, Colonel, USAF
Director, Equipment Engineering

AIR FORCE/56780/17 March 1980 — 400

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ASD-TR-78-48	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE CONFIGURATION MANAGEMENT, One of the Software Acquisition Engineering Guidebook Series		5. TYPE OF REPORT & PERIOD COVERED Final Rept.
		6. PERFORMING ORG. REPORT NUMBER D180-24727-1
7. AUTHOR(s) M. P. Kress		8. CONTRACT OR GRANT NUMBER(s) F33657-76-C-0723
9. PERFORMING ORGANIZATION NAME AND ADDRESS Boeing Aerospace Company P.O. Box 3999 Seattle, Washington 98124		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE64740F Project 2238
11. CONTROLLING OFFICE NAME AND ADDRESS HQ ASD/ENE Wright-Patterson AFB, OH 45433		12. REPORT DATE 2 January 1979
		13. NUMBER OF PAGES 72
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release, Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Acquisition, Acquisition Engineering, Configuration Control, Configuration Management, Configuration Accounting, Allocated Baseline, Certification, Computer Program Configuration Item, Control Software, Software, Validation, Verification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report is one of a series of guidebooks whose purpose is to assist Air Force Program Office Personnel and other USAF acquisition engineers in the acquisition engineering of software for Automatic Test Equipment and Training Simulators. This guidebook provides guidance in the preparation, imposition and enforcement of software configuration management requirements and recommended procedures.

DD FORM 1 JAN 73 **1473** EDITION OF 1 NOV 65 IS OBSOLETE

FOREWORD

This guidebook is one in a series prepared for the Aeronautical Systems Division (ASD/EN) of the Air Force for Automatic Test Equipment (ATE) and trainer simulator software acquisition. It provides guidance to Air Force system acquisition personnel in the preparation, imposition, and enforcement of software configuration management requirements for ATE and Training simulator software through all phase of system acquisition.

This guidebook is one of a series intended to assist the Air Force Program Office and engineering personnel in software acquisition engineering for automatic test equipment and training simulators. Titles of other guidebooks in the series are listed in the introduction. These guidebooks will be revised periodically to reflect changes in software acquisition policies and feedback from users.

This guidebook reflects an interpretation of DOD directives, regulations and specifications which were current at the time of guidebook authorship. Since subsequent changes to the command media may invalidate such interpretations the reader should also consult applicable government documents representing authorized software acquisition engineering processes.

This guidebook contains alternative recommendations concerning methods for cost-effective software acquisition. The intent is that the reader determine the degree of applicability of any alternative based on specific requirements of the *software acquisition with which he is concerned.* Hence the guidebook should only be implemented as advisory rather than as mandatory or directive in nature.

This guidebook was prepared by the Boeing Aerospace Company.

ACCESSION for		
NTIS	White Section	☑
DDC	Buff Section	☐
UNANNOUNCED		☐
JUSTIFICATION		
BY		
DISTRIBUTION/AVAILABILITY CODES		
Dist.	AVAIL. and/or SPECIAL	
A		

This Software Acquisition Engineering Guidebook is one of a series
prepared for Aeronautical Systems Division, Air Force Systems Command,
Wright-Patterson AFB OH 45433. Inquiries regarding guidebook content
should be sent to ASD/ENE, Wright-Patterson AFB OH 45433. The following
list presents the technical report numbers and titles of the entire
Software Acquisition Engineering Guidebook Series. Additional copies of
this guidebook or any other in the series may be ordered from the Defense
Documentation Center, Cameron Station, Alexandria VA 22314.

TABLE OF CONTENTS

LIST OF FIGURES

Section 1.0 INTRODUCTION

Software Configuration Management (SCM), is a unique and distinct discipline essential to the development, delivery and maintenance of contractually compliant software products. Unlike software in earlier systems, software today is being acquired as a product, rather than as data. This is being done in recognition of poorly defined software requirements, poorly planned resources, missed schedules, poor documentation, excessive cost and post delivery problems. In procuring software as a product, provisions are made for documentation, review and verification milestones very similar to those which occur in hardware development. When planned early in the conceptual stage of system acquisition, SCM provides for:

   a. Establishment and maintenance of documentation systems for product identification.

   b. Efficient methods of defining, handling and tracking changes to software products.

   c. Verification and accounting for as-built product configuration and change incorporation.

   d. Periodic reviews and audits of evolving product design.

   e. Verification of ultimate physical and functional configuration.

SCM complements software quality assurance, and other engineering management disciplines in providing for delivery of quality software.

1.1 PURPOSE

It is the specific purpose of this guidebook to present the general principles of software configuration management interpretive of government policy which apply to Automatic Test Equipment (ATE) and Training Simulator (TS) system acquisitions. This guidebook is written to provide insight into some of the pitfalls encountered in configuration management on these systems followed by recommendations for circumventing those pitfalls. It assists the AF project office in developing and imposing cost effective SCM requirements in Requests for Proposal (RFP) and contracting documentation. It also assists system acquisition planners in source selection activities, proposal evaluations, and contractor surveys and audits. During the validation and full scale development phases, this guidebook provides guidance for the review and approval of contractor's Configuration Management Plan (CMP) and Computer Program Development Plan (CPDP). Finally the guidebook provides an overview of the interrelationships of the Air Force, contractor and system vendor throughout the acquisition process.

1.2 SCOPE

This is one of a series of guidebooks related to the Software Acquisition Engineering (SAE) process for TS and ATE ground based systems. The SAE guidebook titles are listed below:

   Software Cost Measuring and Reporting
   Requirements Specification
   Contracting for Software Acquisition
   Statement of Work (SOW) and Requests
     for Proposal (RFP)
   Regulations, Specification and
     Standards
   Measuring and Reporting Software
     Status
   Computer Program Documentation
     Requirements
   Software Quality Assurance
   Verification
   Validation and Certification
   Computer Program Maintenance
   Software Configuration Management
   Reviews and Audits
   Management Reporting by the Software
     Director

1

This guidebook covers generalized concepts and principles supplemented by considerations unique to ATE and TS software configuration management.

The prime areas to be covered are:

   a. Planning and organizing for SCM

   b. Software specifications and end item selection

   c. Baseline definition and change control

   d. Reviews and audits

The guidebook describes the interaction of the Air Force, contractor and system vendor throughout the acquisition process. It further discusses the complementary nature of other contractor functional organizations (i.e. QA, Engineering, Procurement) in the software development process.

1.3   TS AND ATE OVERVIEW

This section provides a brief sketch of TS and ATE system characteristics, including the function of the software associated with each.

1.3.1   TS SYSTEM CHARACTERISTICS

The TS system is a combination of a specialized hardware, computing equipment, and software designed to provide a synthetic flight and/or tactics environment in which aircrews learn, develop and improve the skills associated with individual and coordinated tasks in specific mission situations. Visual, aural, and/or motion systems may be included. Figure 1.3-1 depicts a typical TS which employs digital processing capability.

The computer system, integral to the crew TS, can consist of one or more general purpose computers. The computing hardware operates with floating point arithmetic and sufficient bit capacity to provide efficient use of a simulator Higher Order Language (HOL). When a multi-processor/multi-computer system is used, it must be designed such that computers can operate simultaneously and are controlled/synchronized by a single program (supervisor/executive). The executive directs program execution and regulates priorities.

The simulator (1) accepts control inputs from the trainee (via crew station controls) or from the instructor operator station; (2) performs a real-time solution of the simulator mathematical model; and (3) provides output responses necessary to accurately represent the static and dynamic behavior of the real world system (within specified tolerance and performance criteria). Since TS consist of interdependent hardware and software, a joint hardware/software development effort is required. As the complexity of TS increases, simulation software continues to grow in complexity, size, and cost. Software costs can and do exceed computer hardware costs in many cases. Therefore, it is imperative that the simulation software acquisition engineering process be subjected to formal system engineering planning and discipline to ensure cost-effective procurement.

1.3.2   ATE System Characteristics

ATE is defined as that ground support system which performs vigorous system tests with minimum manual intervention. ATE is used in place of manual devices because it is more cost effective, provides required repeatability, or repair of the item being tested requires the speed which only an automatic tester can achieve.

Figure 1.3-2 shows the typical components of an ATE system. Note that there are both hardware and software elements involved. Most of the elements shown in the figure will be found in the majority of ATE systems although the packaging and interface design may vary between specific systems.

2
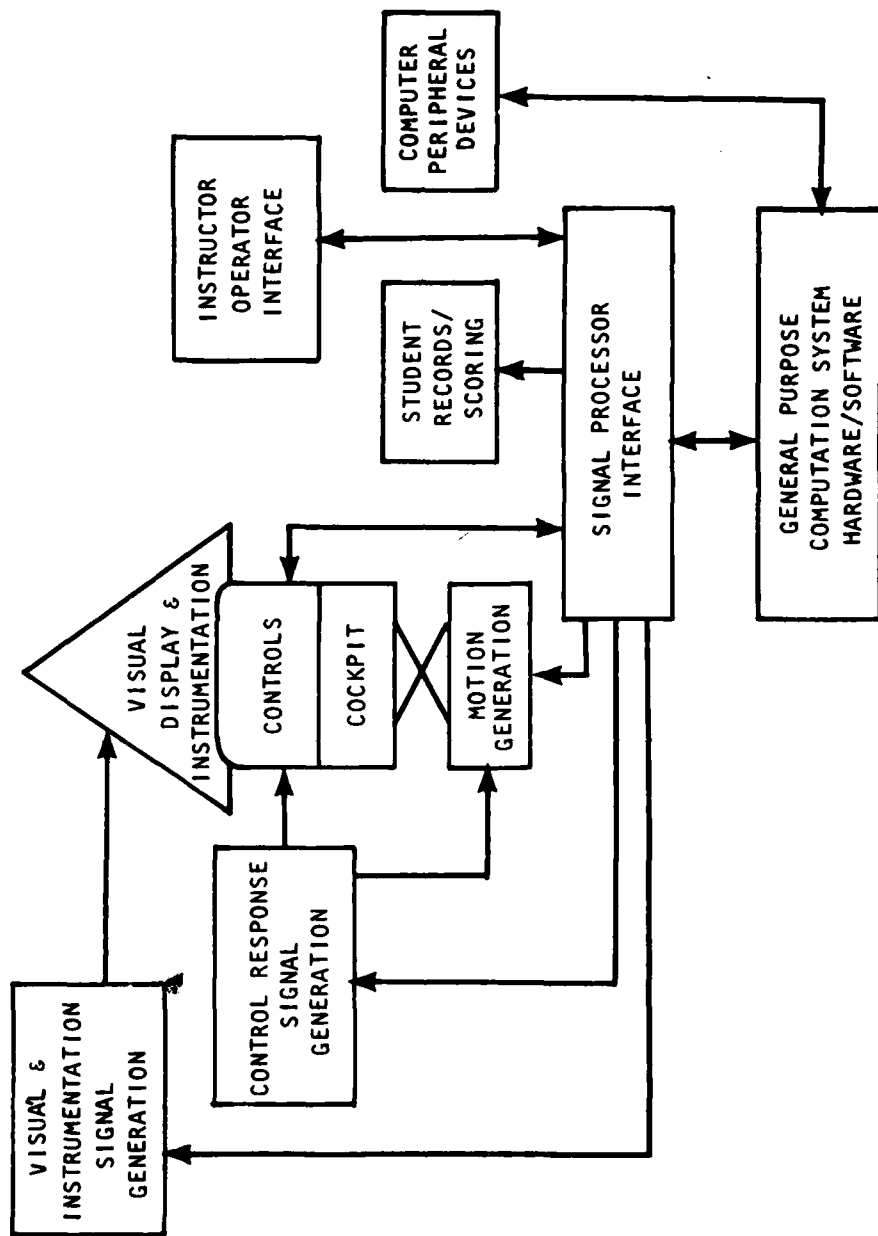
Figure 1.3-1. Typical Crew Training Simulator
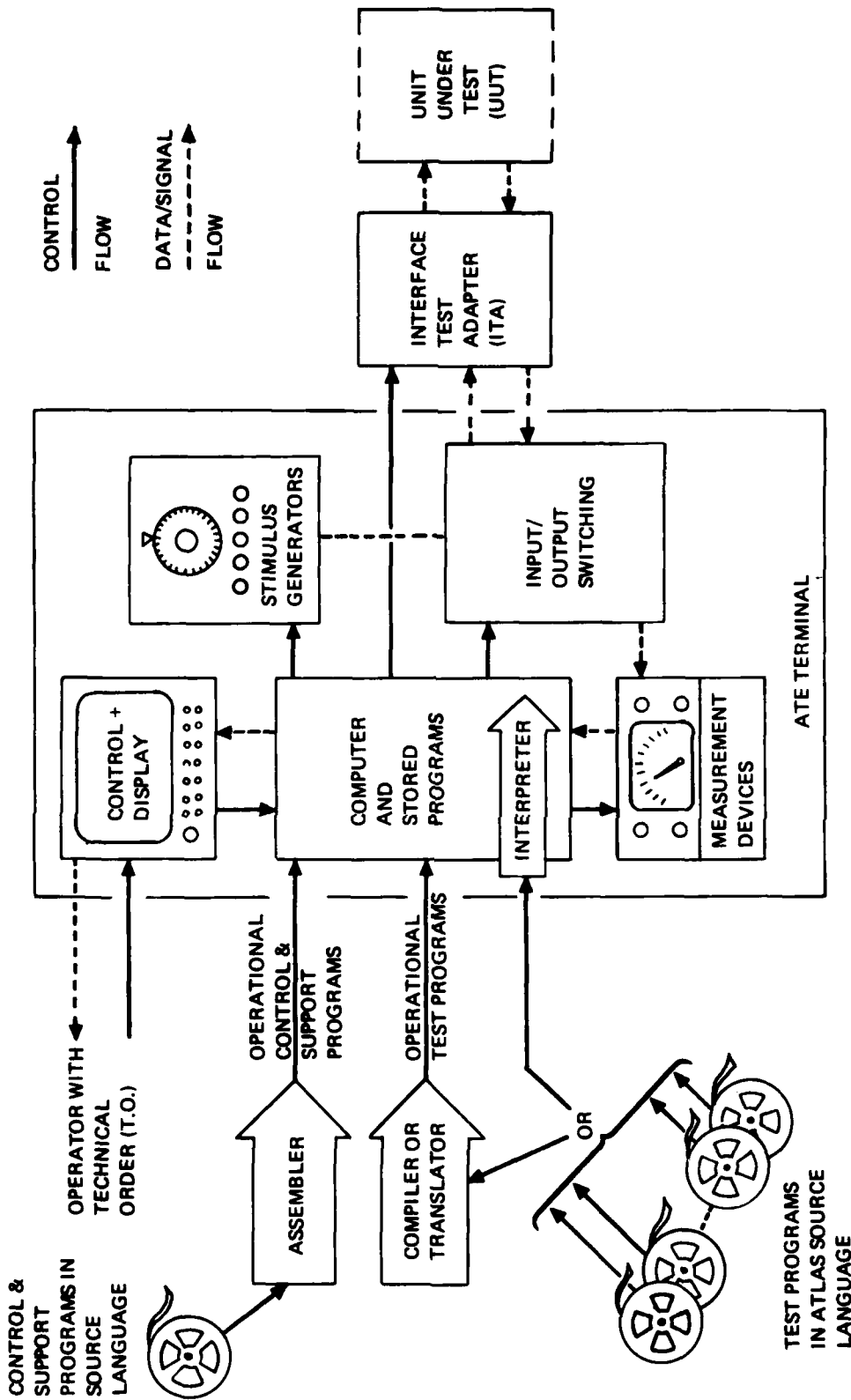
Figure 1.3-2. Typical ATE Configuration

4

The controls and displays section consists of the computer peripheral devices such as control panels, magnetic tape cassettes or disks, a cathode ray tube (CRT), keyboard, and small printer. The computer (normally a minicomputer), as controlled by software, operates the peripheral devices; switches test stimuli on and off; and measures responses of the Unit Under Test (UUT) (comparing to predetermined values). The operator maintains supervisory control of the testing process through the peripherals. However, his interaction is usually minimal since, by definition, the automatic test feature was selected in preference to an operator-controlled test system. ATE is normally designed to accomodate testing several different articles of system equipment (normally one at a time). The maintenance level being supported by the ATE is determined by logistics systems analysis. The importance of the software portion of the ATE system should not be minimized since both the application of the test stimuli and the measurement of the result are achieved via software. Arbitrary function generation and complicated wave analysis can also be accomplished by software and is becoming more prevalent in ATE systems. The cost of ATE software is a significan component of total ATE costs and design trades can be performed to minimize ATE life cycle costs.

1.4 GUIDEBOOK ORGANIZATION

The guidebook is organized as follows. Section 1.0 is introductory. Section 2.0 identifies government documents applicable to ATE and TS software configuration management. Section 3.0 discusses planning for configuration management including resources, facilities and scheduling estimations. Section 4.0 addresses product identification, the position of software components within the system component hierarchy and the sub-structure of software components. Section 5.0 discusses baseline definition and management. Section 6.0 describes change management including change classification, approval, accounting and verification. Section 7.0 describes the functions and management of a typical Computer Program Library (CPL). Section 8.0 discusses formal design reviews and configuration audits of the Computer Program Configuration Item (CPCI). Sections 9.0 through 13.0 contain a bibliography, topic vs government specification cross-reference index, glossary of terms, list of abbreviations and acronyms and a subject index.

Section 2.0 APPLICABLE DOCUMENTS

The following are the major documents which apply to the area of software configuration management.

AFSCM375-7, Systems Management, Configuration Management for Systems, Equipment, Munitions and Computer Programs

MIL-STD-480, Configuration Control-Engineering Changes, Deviations and Waivers

MIL-STD-482, Configuration Status Accounting, Data Elements and Related Features

MIL-STD-483, Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs

MIL-STD-490, Specification Practices

MIL-S-83490, Specifications, Types and Forms

FED-STD-No. 5, Standard Guide for Preparation of Proposed Item Logistics Data Records and Proposed Item Identifications by Government Suppliers

DOD Directive 5010.19, Configuration Management

DOD Directive 5010.21, Configuration Management Implementation Guidance

MIL-STD-499, Systems Engineering Management

AFM66-1, Maintenance Management

AFR65-3, Configuration Management

MIL-STD-1521A, Technical Reviews and Audits for Systems, Equipments, and Computer Programs

MIL-S-52779 (AD), Software Quality Assurance Program Requirements

AFR 800-14, Management of Computer Resources in Systems

DOD Directive 5000.29, Management of Computer Resources in Major Defense Systems

SSD-Exhibit 61-47B, Computer Program Subsystem Development Milestones

7

Section 3.0   PLANNING FOR SOFTWARE CONFIGURATION MANAGEMENT

There are several key considerations which USAF and contractor personnel charged with Software Configuration Management (SCM) must consider early in system acquisition planning. Although ATE and TS software have inherent differences in structure and development philosophy, certain SCM planning elements are common to both systems. Some of these elements are:

   a. definition of software entities

   b. specification of SCM requirements

   c. organizing for SCM

   d. planning for required resources and facilities

The following paragraphs describe how provisions for SCM are established in the early planning stages of software development through discussion of these elements.

3.1   DEFINITION OF SOFTWARE ENTITIES

DOD Directive 5000.29, Management of Computer Resources for Major Defense Systems, in declaring that software products must be developed as configuration items, provides a very definite approach to SCM. In effect, all software entities designated as computer program configuration items (CPCI's) must be developed with controls similar to hardware. A CPCI is, in general terms, a program or aggregate of related programs which satisfy an end function and are specifically designated by the controlling agency for configuration management. This definition does not levy rigorous restrictions on software entities selected as CPCIs.

CPCI selection is an important activity critical not only to successful configuration management, but to technical soundness and manageability. Considerations affecting the choice of CPCI's are

discussed further in paragraph 4.1. However, it is important that planners for SCM (both USAF and contractor) realize that CPCI selection can create difficulties if, for example, too many functions are integrated into one CPCI. If development schedules, language mixes, facilities etc. are not compatible among the components comprising the CPCI, serious SCM problems are encountered. Accordingly SCM planners should become involved in early system design review activities to insure the selection of manageable CPCIs.

3.2   SPECIFYING SCM REQUIREMENTS

There are three major vehicles within the RFP for specifying SCM requirements.

   a. Data Item Descriptions (DIDs) for software documentation

   b. Configuration Management Plan (CMP)

   c. Computer Program Development Plan (CPDP)

Documentation requirements for ATE and TS systems are thoroughly discussed in the "Computer Program Documentation Requirements" guidebook. They are imposed on the contractor as DID's and in most cases they are standard in content and format. In some cases they need to be tailored to meet the requirements and unique aspects of ATE and TS software. For example, the existing DID for preparation of Test Requirements Documents (TRD) (DI-T-3734), references MIL-STD-1519 which is regarded as unsuitable in many cases for ATE software. At a recent joint services conference on ATE (See Bibliography ref. 3), it was concluded that MIL-STD-1519 is too stringent and expensive to comply with, and hence has been watered down in many contracts. (A sub-committee of this conference recommended that a more pragmatic specification common to the tri-services be generated for ATE.) Planners

9

for SCM should therefore examine all of the documentation requirements for software from system specifications through Part II specifications; since these specifications form a population of documentation comprising software baselines against which configuration control must be maintained.

The second major RFP element communicating SCM requirements, is the CMP written for the entire system (hardware and software). This plan provides for overall management of the system components. SCM requirements in their fundamental form are usually specified herein as an addendum to the basic plan. Details of SCM control policy are then expanded in the CPDP.

AF planners should insure that software requirements levied within the CMP are appropriate for all software products involved. For example, in some contracts, a portion of the hardware and software may be supplied as Government Furnished Property (GFP) with existent drawings in a certain format. This format may not be compatible with new build drawings. For software, the integration of two or more CPCI components of differing configuration formats can pose difficult control problems. Altered item drawings are usually required to maintain adequate control. The CMP therefore should address:

a. Basic product identification and baseline definitions.

b. Drawing and documentation requirements.

c. Schedules for Design Reviews and Audits.

d. Classes of Change Control and Effectivity Points.

e. Change Accountability Methods.

f. Unique software configuration management requirements.

The third major RFP element which levies SCM requirements is the CPDP. The CPDP is prepared in accordance with AFR-800-14, Vol II, para. 3-9, by the contractor and submitted with his proposal. The CPDP is the detailed road map for software development and is the prime vehicle for guiding SCM throughout the CPCI development process (see fig. 3.2-1). The CPDP should address the following:

a. The organization, responsibilities, and structure of the group(s) that will be designing, producing, and testing all computer programs.

b. The management and technical controls that will be used during the development, including controls for ensuring that all performance and design requirements have been implemented.

c. The methodology for ensuring satisfactory design and testing, including quality assurance (QA).

NOTE

The requirements of MIL-S-52779, software *QA Program Requirements* are gaining increasing prominence in new weapon systems contracts for both prime and support equipment. The requirements of this specification overlap to a large degree with the objectives of the CPDP. Sometimes the requirement for a software QA plan is levied. In such cases, the CPDP and the software QA plan should be prepared in close coordination because of the inherent similarity of objectives.

d. The development schedule for each computer program configuration item and proposed program milestone review points.

e. The procedure for monitoring and reporting the status of computer program development.

10

Figure 3.2-1. Software Development Process

11

f. The resources required to support the development and test of computer programs. Special simulation, data reduction, or utility tools that are planned for use in the development of computer programs should be identified.

g. The general procedures for reporting, monitoring, and resolving computer program errors and deficiencies during development and testing.

h. The methods and procedures for collecting, analyzing, monitoring, and reporting on the timing of time-critical computer programs.

i. The management of computer program development masters, data bases, and associated documentation including its relationship to the CMP.

j. Guidelines and checkpoints for ensuring future computer program growth, modularity, and ease of modification.

k. The approach for developing computer program documentation.

l. Training requirements and associated equipment for the deployment phase.

m. Engineering practices to include: standards, conventions, procedures, and rules for program design; program structures and conventions; display and logic standards; input/output signal standards; and other disciplines affecting development.

n. Security controls and requirements.

o. Simulation techniques and tasks.

A good CPDP should be prepared in the following format:

I.   Introductory description of policy and objectives

II. Scope - definition of software items under CPDP control

III. Description of general procedures and techniques applicable to all phases of development

IV. Narrative descriptions of operations, control methods, documentation products objectives and review milestones by phase of software development. Phases to be addressed are

(1) Analysis
(2) Design
(3) Code and checkout
(4) Test and Integration
(5) Installation
(6) Operation and support

Each subsection should describe clearly the milestones which demarcate a change in development phase, involvement of AF and management in status reviews, documentation products released at the completion of each phase and the approvals required. See Figure 3.2-2.

V.   Appendices for special procedures, definitions, matrices etc.

A well written CPDP is difficult to prepare because it requires a certain insight into factors difficult to predict at the time it is written. In ATE, for example, factors such as specification completeness and accuracy, program development station availability, human resources, volume of changes, costs, schedules, operating system capabilities, etc. all affect the contractor's ability to follow the CPDP as originally written. If prepared with enough flexibility of method to accomodate these unknowns, while maintaining minimum requirements for disciplined transition from one phase to the next, the CPDP will prove to be a valuable SCM tool.

PRODUCTS OF THIS PHASE ARE

SDR	PDR	CDR		FCA/PCA	
PRELIM. PROD. SPEC. TEST PLAN PREL. INTERFACE DOC. FLOW DIAGRAMS (PREL.)	PROD. SPEC. (DRAFT) INTERFACE DOC. TEST PLAN	TEST PROCEDURES USER'S MANUAL PROG. LISTINGS	VERSION DESCRIPTION DOC. CONFIG. INDEX SOFTWARE VAL. PRO.	TEST REPORTS FINAL VDD FINAL IDD	

ANALYSIS — DESIGN — CODE/DEBUG — TEST/INTEGRATE — INSTALLATION — OPERATION & SUPPORT

ANALYSIS:
- FUNCTIONAL FLOW CHARTS
- DETAIL FLOW CHARTS

DESIGN:
- ANALYZE TRD DATA
- PRELIMINARY TEST PROGRAM
- PRELIMINARY ADAPTER DESIGN

CODE/DEBUG:
- PREP. PRELIMINARY TEST PROCEDURES
- CODE COMPILE DEBUG
- UPDATE PRELIMINARY PROG. & ADAPT DESIGNS

TEST/INTEGRATE:
- MODULE/INTERMODULE COMPATIBILITY TESTS
- ADAPTER TESTS
- PRELIM. PROD. SPECIFICATION PART II

INSTALLATION:
- SOFTWARE & SYSTEM VALIDATION TESTS
- FINAL PRODUCT SPEC.

TO CUSTOMER

Figure 3.2-2. Typical CPDP Flow Plan - ATE

3.3 ORGANIZATION AND RESPONSIBILITIES

Another important SCM planning consideration is that of organizing for SCM. SCM usually requires the time-phased division of responsbilities, in order to achieve efficient control during early development phases, without undue constraint, while insuring formal control during official validation or acceptance testing phases. Ideally a separate contractor SCM function should be established to oversee and manage software configuration control throughout all phases. However, SCM depends on other organizations at various times to meet specific SCM objectives. For example, during pre System Design Review (SDR) phases, systems engineering organizations must maintain preliminary software requirements documents; and test design organizations must maintain Line Replaceable Unit (LRU) acceptance test procedures from which TRD's must be prepared and maintained. During preliminary program design, code and debug, software engineering must maintain source programs, functional and detail flow charts and narrative descriptions suitable for eventual incorporation into CPCI Part II or equivalent specifications. Finally, during validation, QA is responsible for assuring the software "as built and as tested" configuration is documented in an accurate, complete and contractually compliant format.

Interlaced among all of these time-phased SCM responsbilities, the SCM organization itself must function as a central focal point for release and maintenance of all forms of SCM materials used during all phases. Such materials are discussed in paragraph 3.4, "Resources and Facilities." Organization for SCM therefore, is summarily depicted in Figure 3.3-1. As part of contractor project command media, these time shared responsibilities should be documented and committed to by the various functional organizations.

3.4 RESOURCES AND FACILITIES

SCM organizations require adequate manpower and physical facilities for storage and management of large volumes of documentation, changes and computer sensible media. Among the materials to be managed for each software and item are:

a. Source code lists

b. Symbolic decks

c. Flow diagrams

d. Performance specifications

e. Interface description documents

f. Compilers, assemblers, link editors - loaders

g. Data base specifications

h. Load maps

i. Object code listings

j. Utility software

k. Changes and associated records

l. Media conversion/duplication facilities and methods

m. Problem reports/status charges

n. Test Procedures, version description documents

o. Design review and audit schedules

Management of the above is typically accomplished by a Computer Program Library (CPL) function and may be staffed by SCM or QA (MIL-S-52779 (AD) requires establishment of a CPL). Improperly estimating required resources will adversely impact SCM objectives. Usually data does not exist from earlier projects to accurately estimate SCM/CPL

14

	Requirements Analysis Phase	Design, Code, Debug Phase	Validation Phase
Systems Engineering/ Test Design	• Maintain sys Req'mt specs • Maintain Prel.TRD's		
Software Engineering		• Maintain Prel source programs • Module design descriptions • Functional Flows • Problem reports	
Quality Assurance	• Audit SCM Controls	• Audit SCM Controls	• Verify release of VDD • Formal change control • Formal discrepancy control • QA master media control • Formal listings
Software Configuration Management	• Maintain official spec. and drawing release system • Manage CPL (Computer Program Library) function • Problem report handling • Code conversion, compilation, assembly		

Figure 3.3-1. Organizational Responsibilities for SCM

resource requirements. Factors to consider in estimating SCM requirements are:

a. Documentation requirements

b. Schedules

c. Number of CPCI's or components

d. Quantity of anticipated changes

e. Quantity and type of computer media

f. Cost limitations

g. Configuration control aids/utilities

h. Reporting requirements

Resource requirements should be reported in manhours/manmonths and physical facility requirements per CPCI. Procedures should be written defining personnel responsibilities within SCM, CPL organization and filing systems, CPL access requirements, problem reporting and statusing methods etc. If the SCM function is well planned and documented, it will prove to be an indispensable resource in the management of computer programs.

Section 5.0 discusses how these functional and physical software descriptions are managed through a series of customer and contractor controlled baselines.

16

Section 4.0   PRODUCT IDENTIFICATION

DOD Directive 5000.29, Management of Computer Resources in Major Defense Systems, in declaring that software shall be treated as a configuration item, has provided for hardware-like developmental controls for software products. By imposing configuration controls early in the analysis phase of system acquisition, the customer assures himself not only of adequate documentation of the finished product, but of receiving a product which will be more likely to fulfill its functional requirements with minimal maintenance due to errors or design shortcomings. Product identification therefore implies not only the CPCI name and number, but the following physical and functional identifiers.

a. Performance requirements identification.

b. Interface requirements definition.

c. Design definition - design representations, source and object code listings and flow diagrams.

d. Physical media identification.

e. Support software requirements identification.

f. Installation and operating and maintenance instructions.

g. Test requirements and acceptance criteria definition.

Unless all of the above are defined for a given CPCI, the product is incompletely identified. Through the various systems of software documentation available today (see the guidebook, "Computer Program Documentation Requirements)," provisions are made for documenting the above product descriptors.

Although Air Force contracts normally assign responsbility for ATE and TS system acquisition to one contractor, actual management and acceptance is done at the configuration item (CI) level. For software, the ordered set of instructions and coded data, in a form suitable for insertion into a computer, designed to accomplish a specific set of functional requirements, is known as a Computer Program Configuration Item (CPCI). The selection of CPCI's for management and control purposes is a task critical to successful software development and configuration management. AFSCM/AFLCM 375-7 provides basic guidance for CI selection. Individual selections however must be the end product of technical and administrative systems engineering analysis. The following section examines factors affecting this selection for ATE and TS CPCI's.

4.1   SYSTEM SPECIFICATIONS

System Specifications for ATE vs. TS are as diverse in content and structure as the functions they serve. Specifications for ATE are written in terms of stimuli, measurement, accuracy and thruput requirements. TS requirements are written in terms of real time simulation requirements. These differences are fully described in the "Requirements Specifications" guidebook. They are discussed here only to provide a starting point for a discussion of baselines and CPCI selections. Although they are diverse in functional objectives, certain considerations common to both affect the selection of CPCI's. Major system level factors which should be considered in CPCI selection are:

a. _Processor_ - Programs designated to operate in a given computer should be separate CPCI's.

b. _Schedule_ - Programs designated for delivery at different times (i.e. operating system software is developed and delivered prior to simulation or LRU test programs) should be separate CPCI's.

17

c. Deployment - Programs designated for supporting a limited intended usage (i.e. LRU test programs designated to support contractor factory acceptance test should be separate from the same program used for intermediate level maintenance) should be separate CPCI's.

d. Contractor - Program development by separate contractors (i.e. LRU programs - vs. - control/support programs) should be separate CPCI's.

CPCI selections are made to optimize technical and administrative control. The totality of CI's and CPCI's satisfy the system specification or system segment specification for the applicable ATE or TS system. It is important that configuration management planners participate in System Requirement Reviews (SRR) and System Design Reviews (SDR) to register their concerns on requirements allocation to CI's and CPCI's from a configuration manageability standpoint. Some typical concerns to register are:

a. Can anticipated change processing flow times support development schedules for the CPCI?

b. Is the classification of change control and its point of effectivity clear and acceptable for each type of CPCI?

c. Are the CPCI's sized for manageability?

d. Are resources available to control configuration of special required support utilities? - (i.e. host computer facilities needed to develop, assemble compile, link edit etc.)

e. Will the distributed architecture nature of modern ATE software present any special configuration maintenance problems? (i.e. programs operating in micro and mini computers remote from the central processor.)

f. Are formal review and audit schedules realistic for all CPCI's.

g. Is there clear distinction between CPCI and non CPCI products.

In order to build manageable CPCI's, the system specifications upon which CPCI development specs are based must be complete and accurate. System engineering analysis and design trade studies should form the bulk of justification for CPCI selections. The foregoing has presented some system level considerations common to ATE and TS software and configuration management. The following paragraph discusses some SCM considerations unique to ATE and TS which affect CPCI selections, and software specifications.

4.2 SOFTWARE SPECIFICATIONS AND CPCI SELECTIONS

The evolution of software specifications is shown typically in Figures 4.2-1 and 4.2-2 for ATE and TS respectively. In reality the actual relationships among requirements, development and product specifications is as diverse as the systems themselves. It is noted in the example of Figure 4.2-1 for instance, that a software system specification exists to govern the performance of four CPCI's. This ensures a uniform approach to development of programs is adhered to, regardless of the originator of the test requirements. The fact that this relationship varies among contracts is inconsequential to SCM. What is essential. however, is that program development specifications are based on CPCI selections which are manageable and on system requirements which are complete and unambiguous.

Figure 4.2-3 is an example of an ATE system configuration in which functional capabilities have been assigned to various "stations" to accomodate a wide range of UUT's destined for multiple user locations. This figure graphically represents all potential configurations of all stations at any time.
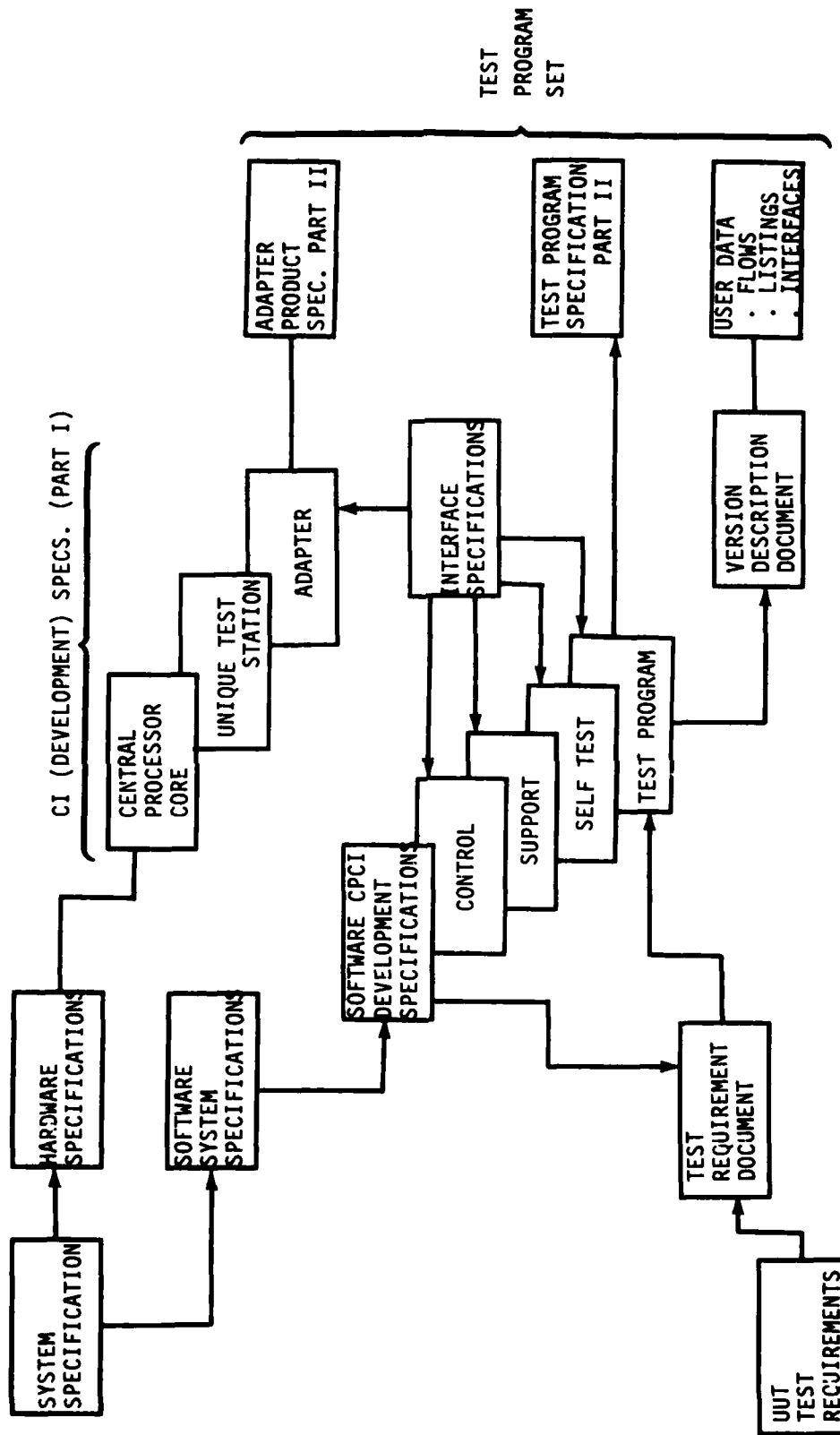
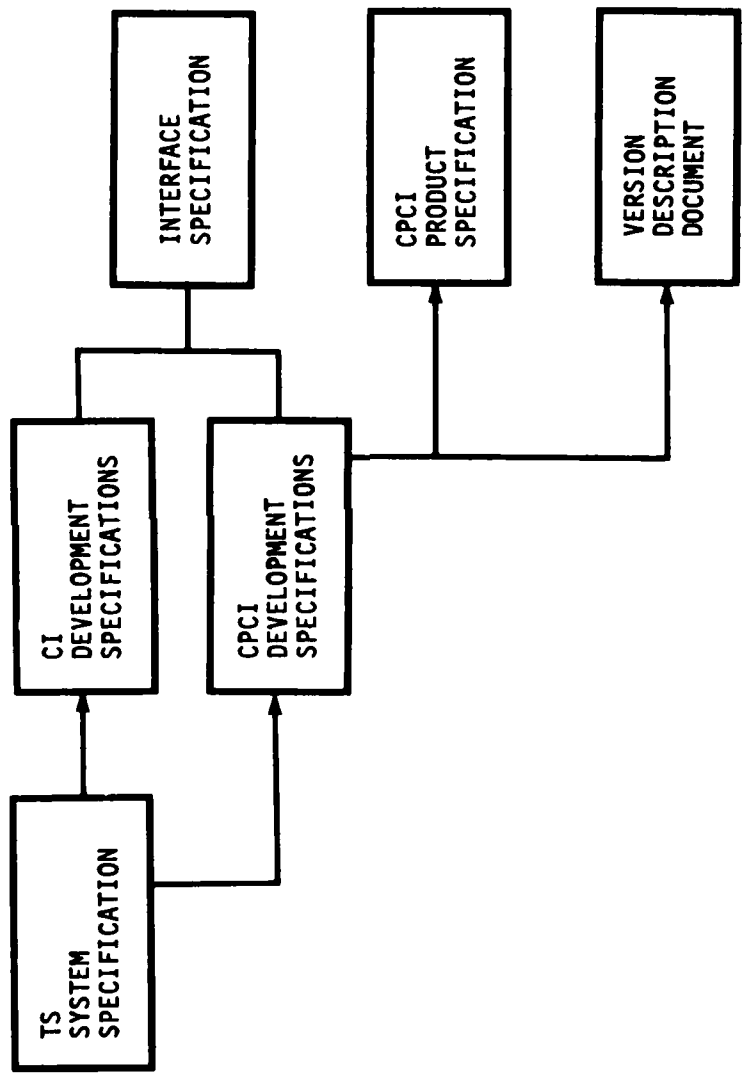Figure 4.2-1. ATE Software Specification Relationship

19

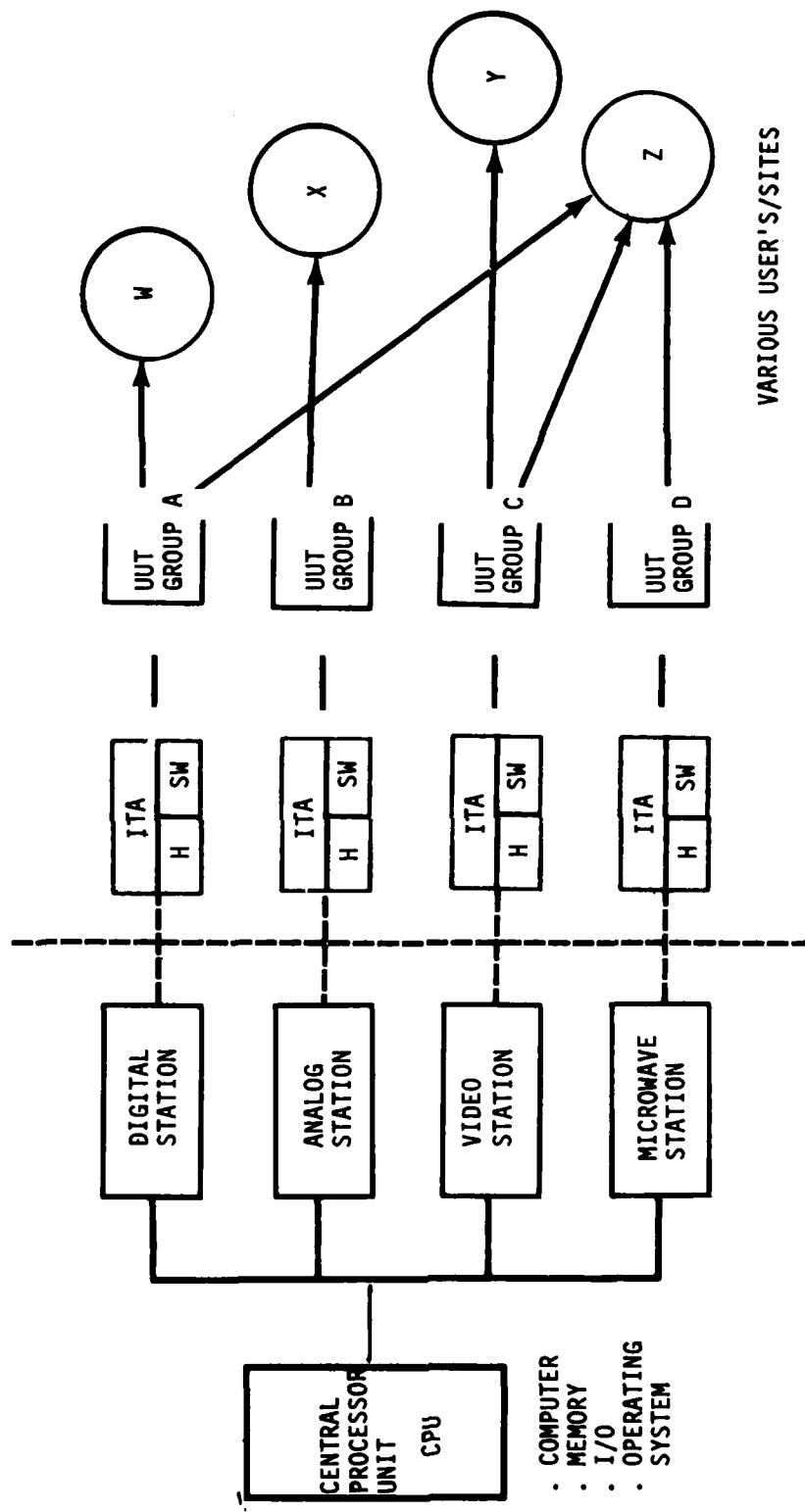Figure 4.2.2. TS Software Specification Relationship

20

Figure 4.2-3. Example of Multiple ATE Configurations/Multiple User's

21

Any given station configuration is comprised of a:

a. Central Processor Unit (CPU)

b. Functionally unique station

c. Interface Test Adapter (ITA)

d. UUT Test Program

Software architecture may be centralized (residing and executing primarily in the CPU) or distributed (executing in micro/mini processors residing in the station or adapter. Adapter circuitry may be active or passive. LRU test programs may or may not be transportable among stations and adapters. Users may be contractors or AF maintenance personnel. It is seen therefore that system configuration control can become complex and unmanageable if rigorous methods of baseline definition and change management are not instituted. To appreciate how test requirements allocations to CPCI's can affect SCM, consider the case shown in Figure 4.2-3.

To preserve system configuration commonality among all users (a design goal for most ATE), any change in test capability required of any one user/site demands that the same change be incorporated in all stations of like configuration. As the quantity of deployed stations of a given configuration increases, the configuration control problem increases. Several possible ways of accomodating such a change are:

a. Incorporate required change in CPU or station hardware/firmware.

b. Incorporate required change into the site unique adapter.

c. Modify the UUT software.

d. Any combination of the above.

Such change implementation is a complex matter. Frequently design trade studies and system engineering analysis are required to determine what capability shall exist in the central processor vs. the station vs. the adapter vs. the software. The resultant decision will affect software and maintainability significantly. Therefore the clarity and completeness of all specifications - system, CI and CPCI are mandatory to effective SCM.

4.3 CPCI/COMPONENTS IDENTIFICATION

Functional requirements of CPCI's should be allocated to subcomponetry in accordance with many of the same technical and administrative ground rules as apply to system functional allocations. The principles of structured programming about which much has been published (see Bibliography references 4 and 5), apply more directly to TS software than to ATE. For ATE test programs, similar principles of modular decomposition apply. UUT test programs are generally a sequentially executed collection of individual tests with little inter-module (or inter test) compatibility to worry about. For TS software the development spec (CPCI Part I) governs the structure of the CPCI design, while the applicable programming standards and naming conventions govern its format. As the subfunctions of a given CPCI are ascertained in some fashion according to designated criteria for CPCI decomposition, these subfunctions are allocated to modules (or test numbers in the case of ATE). The following is an example of a TS software module identification scheme designed to achieve maximum information from the module part number.

123-45678-42203
— Module version number
— Module number
— Major system level revision number
— CPCI designator (i.e. audio simulator)
— System designator (weapon system X)

22
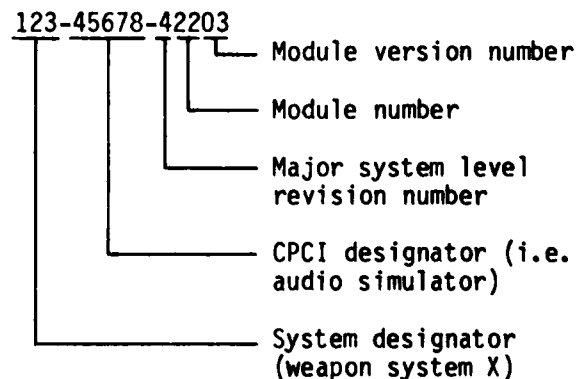
The above 15 character part number fully identifies a given CPCI module. The module can be described from this part number as "the 3rd version of module No. 22, part of the 4th major system revision to CPCI 45678 for system 123.

Total traceability of the module to its parent CPCI and system configurations can be maintained with this identification system. Each module can be designed and changed independently of other modules while maintaining traceability to the parent CPCI version. Such control is necessary to distinguish modules during concurrent development of different CPCI versions. Each module part number then appears on an indentured parts list for the specific CPCI.

A version description document (VDD) (Figure 4.3-1) is used to group such modules for a given CPCI version. It is seen from this figure that a given CPCI version, (123-45678-401) is configured as a collection of

   a. Functional modules

   b. Physical media

   c. Support programs necessary to assemble, compile etc.

   d. Labelling instructions

   e. Acceptance test requirements

   f. Object program listings

   g. Operating instructions

Everything required to build, identify, test, install and operate a given CPCI version is called out on the VDD in indentured parts list format. In the example given, it is seen how the module revisions are progressively updated for the next version of the CPCI. Control is maintained in this way by preserving the configuration of previous versions upon which later versions are built.

Safeguards are therefore provided against the "what-did-we-build-last time?" syndrome frequently encountered in uncontrolled software development. Several part number relationships are logically evident in this scheme.

   a. CPCI version -401 contains module versions which are predominantly "-01" (i.e. -42201).

   b. Major CPCI revisions (i.e. -501 vs. -401) contain corresponding major modular redesigns (5XXXX series modules)

   c. Module or component specification dash numbers correspond to the module numbers (i.e. module -42201) is described by design spec XXX-XXXXX-22)

As the discussion of baseline and change control mechanisms evolve in subsequent sections of this guidebook, the value of these inter-relationships will become more evident.

QTY REQ'D 502	QTY REQ'D 501	QTY REQ'D 403	QTY REQ'D 402	QTY REQ'D 401	PART NUMBER	MODULE/COMPONENT NAME	COMPONENT SPECIFICATION	REVISION
				1	40101	EXECUTIVE	XXX-XXXXX-1	A
				1	40201	NAVIGATION SIM.	XXX-XXXXX-2	A
				1	40301	VISUAL DISPLAY SIM.	XXX-XXXXX-3	A
				1	40401	MOTION SIM.	XXX-XXXXX-4	A
				1	42201	AUDIO SIM.	XXX-XXXXX-22	A
				1	XXXX-1	MAGNETIC TAPE	MATERIAL STD -XYZ	
				1	YYY	LABEL	NNN-1	
				*	L00401	LISTING		
				*	ATP-1	ACCEPTANCE TEST PRO.	123-78910-1	-
				*	QQQ-1	OPERATION/INSTALL. INST.	123-32165-1	A
				*	SUP-1	SUPPORT SOFTWARE	123-87654-1	B
1					50102	EXECUTIVE	XXX-XXXXX-1	B
1					50202	NAVIGATION SIM.	XXX-XXXXX-2	B
1					50303	VISUAL DISPLAY SIM.	XXX-XXXXX-3	B
				1-CONTAINED WITHIN		*- APPLICABLE TO DESIGNATED OPCI	Page 20 of 123-45678 (CPCI NUMBER)	

Figure 4.3-1. Version Description Document

24

Section 5.0   BASELINE MANAGEMENT

A baseline, in general, is a documented technical description which becomes a point within a development process against which changes can be proposed, evaluated and incorporated. Three configuration baselines for ATE and TS software are described in this section as a function of the development cycle. Baselines are employed throughout the software life cycle to ensure an orderly transition from one major commitment point to the next. A configuration identification document or a set of such documents, formally designated and fixed at a specific time during a CI/CPCI's life cycle establishes a baseline. Baselines, plus approved changes from those baselines, constitute the current configuration identification. This section will discuss the three formal AF recognized baselines for ATE and TS software, as well as internal contractor controlled baselines.

5.1   TYPES OF BASELINES

The three formally recognized baselines applicable to all CI or CPCI items are:

a. <u>Functional Configuration Baseline</u>. The current approved or conditionally approved technical documentation for a configuration item as set forth in specifications and documents which prescribes: all necessary functional characteristics; the tests required to demonstrate achievement of specified functional characteristics; the necessary interface characteristics; key functional characteristics and key lower level CPCI's; and design constraints.

b. <u>Allocated Configuration Baseline</u>. The current approved performance-oriented specifications governing the development of CPCI's that are part of a higher level configuration item, in which each specification: defines the functional characteristics that are allocated from those of the higher level configuration item; establishes the tests required to demonstrate achieve-

ment of its allocated functional characteristics; delineates necessary interface requirements with other associated CI's/CPCI's; and establishes design constraints.

c. <u>Product   Configuration   Baseline</u>. The current approved technical documentation which: defines the configuration of a CPCI during the production, operation, maintenance and logistic support phases of its life cycle; defines all necessary form, fit, and function characteristics of a CPCI; defines the selected functional characteristics designated for acceptance testing; and defines the acceptance tests.

These baselines are shown on Figure 5.1-1 as external baselines; i.e., configuration control is external to the contractor organization; changes are subject to customer approval. Software configuration control is normally initiated with the allocated baseline, which is established by USAF acceptance and approval of the computer program development specifications at SDR; that baseline is used as the point of reference for formal configuration control (see Paragraph 11.6) until the product baseline is established upon customer acceptance of the computer program product specification at Functional Configuration Audit (FCA)/Physical Configuration Audit (PCA).

Internal baselines (not subject to USAF control) should be used by the contractor to control the software design during the development of the computer program product specification. This covers the period between the allocated and product baselines as shown on Figure 5.1-1. Two such internal baselines should be established for the software design. The internal baseline of the basic design for each computer program should be established at the Preliminary Design Review (PDR) by release of a partial draft of the computer program specifications. The internal baseline for the detailed (module) design should be established at the Critical Design
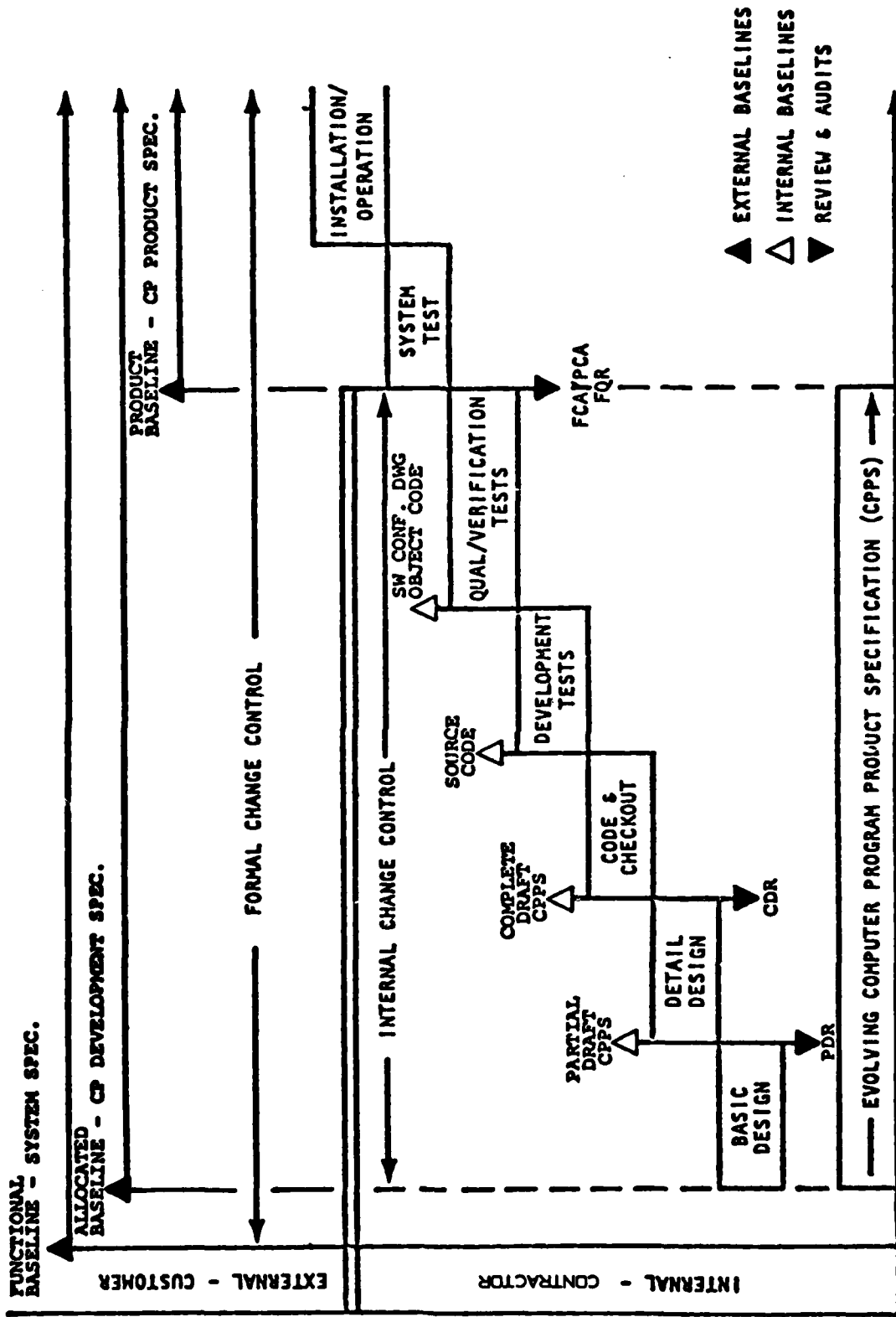
25

Figure 5.1-1. Baselines and Controls

26

Review (CDR) by release of the completed draft of the computer program product specifications. The internal baseline for the detailed (module) design should be established at CDR by release of the completed draft of the computer program product specification less listings. Internal baselines for testing of the computer program should be established by release of a computer program configuration drawing, release of the source code to a CPL, and release of the object code to quality assurance for control during test conduct.

The foregoing is a generalized description of baseline control for all configuration items. It is intended to demonstrate the three basic developmental milestones for configuration control. Because of certain fundamental differences between ATE and TS software, each must be treated differently with respect to baseline control.

5.2 BASELINE CONTROL VARIATIONS- ATE VS TS

The following subparagraphs discuss some of the variations in baseline control required to accommodate some inherent differences between ATE and TS software.

5.2.1 Intermodule Dependence

Perhaps the most significant difference between ATE and TS software relative to baseline control is that simulator software is designed to simulate a real time situation requiring modular interplay, whereas ATE programs, (i.e. UUT programs) execute test commands sequentially in a batch processing mode. TS programs, therefore, require both "inter-" and "intra-" module compatibility and control documentation.

Good design and configuration control of inter-module interface signal processing (timing and sizing of intermodule data traffic) is an essential requirement for TS software. On the other hand, ATE software, although it too has interface software compatibility requirements (among

control, support and test components) is comprised mostly of test programs which are independent, sequentially executed test statements. This difference is shown diagramatically in Figure 5.2-1. It is seen here that during deployment, intermodule compatibility control is not as much a factor for ATE software while, it is a significant factor for TS software.

Interface control, therefore, must be established by interface control documentation which defines detailed interface requirements. This becomes a portion of the allocated and product baseline documentation in appropriate CPCI specifications. Interface control documentation establishes specific functional or physical relationships that must exist between computer programs and computers, other system equipment and other CPCI's; and between computer program components of CPCI's to achieve integration and compatibility. The development and verification of interface documentation and its inclusion in the appropriate specifications are essential prerequisites to the completion of baseline definitions.

5.2.2 Distributed Processing

Another significant difference between ATE and TS software which affects baseline control is the distributed architecture nature of ATE software. The proliferation of the mini and micro computer has changed the structure of ATE system software from one characterized by central processing to one of distributed processing (i.e. more functions being performed by the remote station rather than by the central computer.) Control of firmware (programmable read only memories) (PROM), is becoming a configuration control ingredient essential to overall ATE software configuration control. Control of software configuration is now more hardware dependent. Mini-functions such as signal generation and analysis are being done by the microprocessor while limit information (pulse amplitude, width, rep rate) is supplied
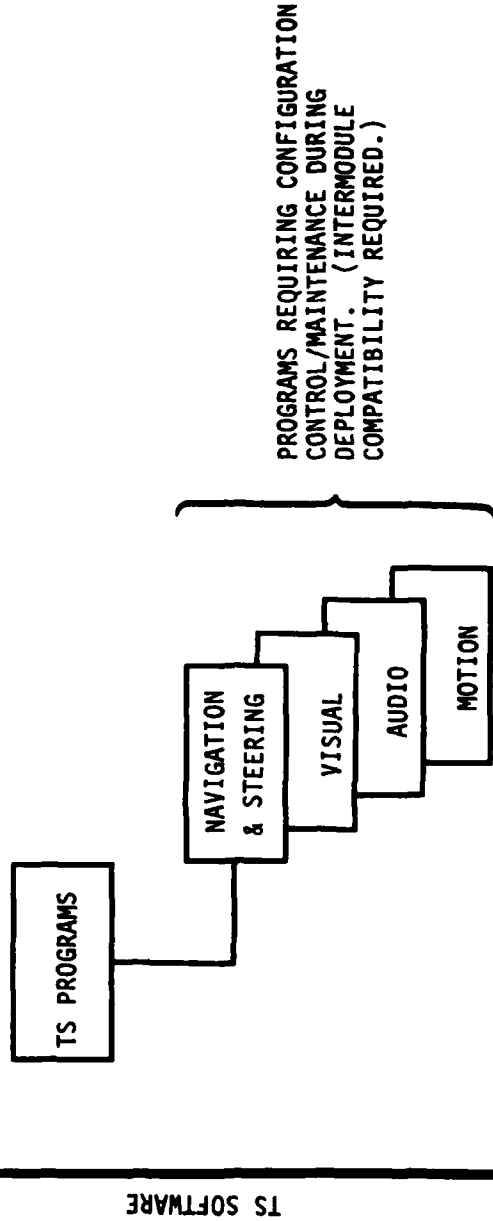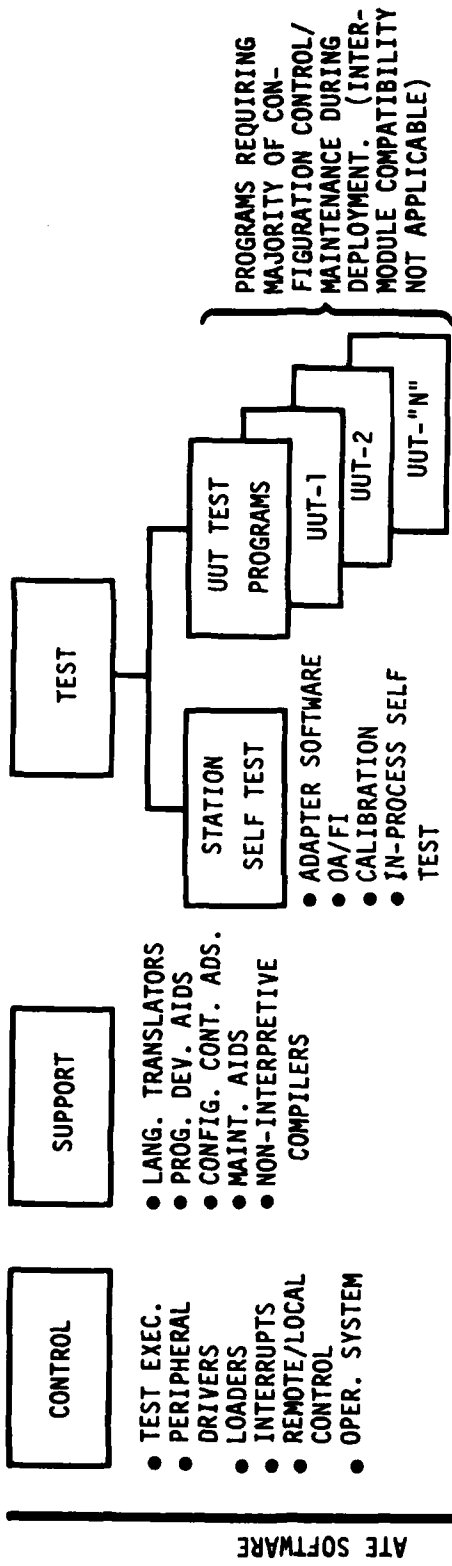
Figure 5.2-1. ATE Versus TS: Configuration Control Variants

28

by the UUT test program. The software configuration controller, therefore must know what programs exists in PROMS as well as in the UUT source program to fully configure the UUT test software.

5.2.3 Change Volume

Another significant factor affecting baseline control differences between ATE and TS programs is the volume of changes. During developmental phases, change volume is variable for both systems, however, during deployment changes to system LRU's affect ATE software differently than TS software. Once deployed, an avionics LRU change would have to be functionally significant before it would affect a simulator program. It would have to alter functional capability of the system before it would require a corresponding change to the simulator program. For ATE software, on the other hand, many of the "as-built" changes require UUT program changes to insure functionality has been restored or to accommodate signal parameter limit changes.

These variants for ATE and TS software impact the methodology developed for baseline control. The resources, skills, problem reporting methods, and change incorporation and verification methods are different as a result of these variants. In the following subsections, we will examine some typical baseline control mechanisms for both types of software in which impact of these variants will become evident.

5.3 BASELINE CONTROL MECHANISM

There are two distinct elements or sub-tasks involved in maintaining baseline control:

a. configuration definition

b. configuration accountability

Configuration definition is a software engineering responsibility and is comprised of baseline definition plus

change definition. Configuration definition is maintained by releasing engineering documents such as Part I, Part II specifications and VDD's, plus approved changes thereto.

Configuration accountability is the process of verifying that the applicable computer sensible media reflects the approved baseline and that changes to that media are incorporated only in authorized versions or at other committed effectivity points.

Since ATE and TS software are so diverse structurally the following subsections address baseline control separately.

5.3.1 ATE Software

ATE software baseline control begins with the system functional baseline. The functional baseline is established by the system specification for the ATE and SDR. At this point, top level system (hardware and software) functional requirements are defined. The system specification defines required capability in terms of:

a. Number and type of UUT's requiring test.

b. Computational capacity as a design goal.

c. Stimuli/measurement parameters, accuracies.

d. Reliability, maintainability goals

e. Diagnostic vs. end-to-end test capabilities.

These are functional requirements for the ATE as a system, irrespective of the selected ATE design.

The functional baseline, in effect, defines the job to be done by the ATE for a specified range of products under test. The population of UUT's categorization, and various logistics support requirements determine the system requirements.

These system requirements are then "allocated" to CI and CPCI's. For example, a given ATE configuration may be allocated into CI's and CPCI's as follows.

a. CPU

b. Digital station

c. Video station

d. Microwave station

e. Servo station

f. System software

    (1) Control

    (2) Support

    (3) Test

g. Communication/navigation

h. Program development station

Each of the above is designated as a CI (or CPCI for software), the requirements for which are defined in a series of development specifications to form an "allocated" baseline for that CPCI/CI. An outline for an ATE system software CPCI development spec is shown in Fig. 5.3-1. The software allocated baseline is herein defined. It is prepared by the ATE contractor during the analysis phase of the ATE computer program development cycle and approved by the Air Force at the CPCI PDR.

The purpose of a development specification needs to be reviewed at this point. A development specification is prepared primarily as a two-way agreement between the Air Force and the development contractor. It is prepared independent of the design approach. It specifies what the software shall do (function), how well it shall do it (performance) and under what conditions (design constraints). In addition, it provides validation requirements that define the scope of the validation program. The

specification is used as the functional and performance baseline for the contractor in developing computer programs and is also used as the baseline on which Air Force acceptance or rejection of the computer program is based.

A statement of computer program requirements, that have been approved by the contractor and the Air Force, is a necessary instrument for a clear understanding of what the contractor will produce and what the Air Force expects. The three basic categories of ATE computer programs present different problems in the need and generation of development specifications. The form and substance of a development specification may vary for each CPCI depending on the category and the degree to which it must be developed. Since ATE software categories are different, they are often designed as separate CPCIs. That implies separate specifications, separate development schedules, separate review schedules and separate validation of requirements that must all be coordinated and eventually "sold" as a unit. This raises the possibility of an ATE software system specification covering all CPCIs, as well as the individual CPCI specifications. Since the CPCIs can and do have separate development schedules, each CPCI would have its own PDR and CDR schedule; e.g., normally the control and support software development precedes that of the test software.

Most of the time, control software and support software are purchased from an ATE vendor as part of a test set or separately from the computer manufacturer. Control software and support software are usually identified as separate CPCIs. When these computer programs are purchased "off-the-shelf," development specifications are not required. Computer vendor documentation is required for computer program maintenance and for the possibility of making changes to the purchased computer programs. The equivalent of a product specification (Part II, MIL-STD-483)

ATE SOFTWARE DEVELOPMENT SPEC OUTLINE

(ALLOCATED BASELINE)

Sect. 3.0 (Part I per MIL-STD-483)
Requirement

3.0	Requirements
3.1	Definitions
3.1.1	Station Test Software
3.1.2	Station Control Software
3.1.3	Station Support Software
3.2.1	Station Test Software
3.2.1.1	UUT Test Software
3.2.1.2	Confidence Test Software
3.2.1.3	Operational Assurance/Fault Isolation
3.2.1.4	In Process Self Test
3.2.1.5	Maintenance Software
3.3.1	Station Control Software
3.3.1.1	Test Control Software
3.3.1.2	Remote Mode Test Control Software
3.3.1.3	Program Development Control Software
3.4.1	Station Support Software
3.4.1.1	Maintainability
3.4.1.2	Operating System
3.4.1.3	File System
3.4.1.4	Loader
3.4.1.5	Operator Interface
3.4.1.6	Batch Processing
3.4.1.7	Language Processors
3.4.1.8	Program Development Aids
3.4.1.9	Interactive Editor
3.4.1.10	Automatic Test Program Generator
3.4.1.11	Configuration Control Aids
3.4.1.12	File System Cataloguing
3.4.1.13	Configuration Verification
3.4.1.14	Media Conversion
3:4.1.15	Maintenance Software

Figure 5.3-1. ATE Software Development Specification Outline

should be obtained from the vendor. Program listings and source code are almost indispensable. When significant additions or changes must be made to the purchased control or support software, a development specification should be written covering the changes to be made, and the interfaces required with the purchased computer programs and the test equipment. The existing computer program (obtained from the vendor) is identified as an interface and the additional software, treated as a CPCI, will be designed, tested, reviewed and controlled accordingly. When control and support software are to be totally developed by the contractor, a complete development specification is required.

There is considerable controversy within the Air Force and contractors as to whether a development specification is applicable for test software. One point of view is that test software is a computer program and must be developed as a CPCI. Therefore, a development specification is required for proper control of the development process. The other point of view is that test software is derived from a TRD, which can be most efficiently written directly in the *Abbreviated Test Language for All Systems* (ATLAS) language by a UUT design engineer, thereby bypassing the need for a development specification or at least regarding the TRD as the development specification.

The point to remember is that test software is dependent not only on the TRD, but on the ATE test set, the ITA and the station resident software as well. It is clear, therefore that the TRD itself written independent of the ATE, ATE software and ITA, is not an adequate substitute for a development specification and cannot provide the design definition and control.

The allocated baseline therefore must be defined by a development specification which addresses.

a. Control software requirements

b. Support software requirements

c. Test software requirements

(1) UUT test software

(2) Self test and calibration

d. Design requirements (ATLAS version(s)

e. Quality Assurance

(1) Configuration Requirements

(2) Functional and performance verification requirements

When approved at PDR, the allocated baseline is authority to proceed into the design phase of ATE software development. TRD's are prepared in accordance with the development specification and the appropriate contractual specifications, such as MIL-STD-1519. However, it was recently concluded by the Joint Services and Industry (see Bibliography ref. 3) that MIL-STD-1519 is too *stringent* and expensive. In practice each branch of the service has had its own interpretation and the current consensus is that more pragmatic specification is required for TRD's.

From the TRD/development specification, work can begin building a "product" baseline. The product baseline is defined by the product specification and defines the as-built design. The product specification consists of three essential parts.

a. Logic flow diagrams

b. Support narrative descriptions

c. Source program listings

Preliminary drafts of the product specification are made available at PDR and completed drafts available at CDR. This specification is maintained under Class II control from PDR until it becomes

deliverable or at PCA at which time it is henceforth maintained under Class I control.

When the control and support software position of ATE software are purchased off the shelf as is the case in most ATE systems available today, the job of maintaining baseline control of the software boils down to maintaining control of the UUT test programs themselves. The UUT program is one element of what is called a Test Program Set (TPS). A TPS, theoretically is everything an operator needs to run a UUT test given an ATE system. A TPS consists of

a. A UUT object code test program

b. Adapter or interface device

c. Operator instructions consisting of supporting data adequate to achieve self maintenance

(1) Flow diagrams, schematics

(2) Source diagrams

(3) Interface diagrams

(4) Test loop diagrams

One of the primary purposes of ATE software configuration control is to provide at time of delivery, complete and accurate software configuration and usage documentation to the user. Too frequently, developers of UUT programs are under pressure to "debug" and deliver with the result that the related documentation suffers. Care must therefore be taken to the SCM group to insure program version updates are accompanied by updates to all affected documentations required by the product specification. This includes

a. New versions of compilers/ interpreters

b. Firmware changes to hardware design

c. Interface descriptions

d. VDD

e. Flow diagram + narratives

f. Part 1 specification (if applicable)

g. Adapter changes

Fortunately, for ATE software, the language and operating system accomplish much of the configuration control job through automated documentation techniques.

For example, consider a UUT program coded in ATLAS. Most ATE systems available today include a complete repertoire of compilation, file management, text editing and configuration control utilities. The system compiler accepts ASCII coded source language statements and uses configuration information output by the ATE and adapter/interface (A/I) processors to generate an object code program. The ATE configuration processor accepts a description of the instrumentation, switching and method of the programming, then outputs files used by the A/I processor and the compiler. The A/I processor accepts a description of signal routing through the A/I between the UUT and the ATE, and data files output by the ATE configuration processor, then outputs files used by the compiler. If the instrumentation, switching, programming method, adapter or interface changes, file changes must be made to the ATE and A/I configuration files so that accurate and up-to-date system configuration information is used by the compiler (see Bibliography ref. 6).

Other ATE system capabilities, normally supplied as part of the vendors operating system, greatly aid the job of maintaining ATE software baseline control. SCM personnel should become familiar with the capabilities of the selected ATE system to effectively utilize the systems capabilities for configuration

control objectives. Additional automated aids to configuration control are (see Bibliography ref. 7):

a. Automatic prevention of unauthorized updates to configuration controlled tapes, e.g., by checking an authorization number in the update request with a corresponding change approval code in the SCM Data Bank.

b. Automatic SCM Data Bank recording of "was-is" data during controlled updates, showing old and new memory contents.

c. Automatic generation of flow charts from the test program coding, guranteeing that the flow charts are up to date with the code, a determination so time-consuming and difficult by visual comparison, that it seldom is performed and then only on a limited sampling basis.

d. Automatic comparison of the configurations of the pre- and post-validated test program, resolution of differences with SCM Data Bank records of approved changes, and isolation of unapproved changes.

e. Automatic generation of integrated configuration lists and other types of configuration status accounting reports.

f. Automatic generation of wire lists.

g. Automatic documentation update and change distribution.

h. Automated revision tracking of source and object code files. The revision code is carried as a field in the file name and automatically implemented by the operating system when the file is accessed for editing.

Utilization of these automated aids together with disciplined manual change processing methods will provide for complete and accurate ATE configuration control. Methods of change processing are discussed in Section 6.0.

5.3.2 TS Software

Baseline control for TS software originates (similar to ATE software) with the functional configuration baseline for the simulator system. As shown in Figure 5.3-1, the TS system specification defines the functions to be performed by the TS without regard to implementation, (hardware or software). When approved by the System Program Office (SPO) and released in the RFP, the contractor prepares a technical proposal describing how each functional requirement will be met. Hardware/software trade studies determine the number of tasks "allocated" to software.

Unlike ATE software the system functions allocated to TS software are not as clear cut. (Refer to the guidebook on "Requirements Specifications," for a discussion of hardware/software trades.) The functions allocated to software are usually undetermined at the time of RFP release. The contractor's proposal identifies the software modules, their interfaces, and functions to be performed by each module. In accordance with DOD Directive 5000.29 and AF Reg. 800-14, software programs must be managed as a CPCI's and consequently a CPCI development spec, (Part 1) should be prepared to form an "allocated" software baseline. Some of the problems associated with TS software requirements baseline control are

a. Untimely requirements changes

b. Excessive MIL-spec design/construction constraints

c. Excessively high fidelity requirements

d. Excessive instructor displays and controls

e. Inexact verification requirements

f. Ambiguity in requirements

Unlike ATE software, a requirement overlooked, misinterpreted or changed without good reason causes significant cost, schedule and configuration control difficulties. Most test software changes can be accommodated relatively easy. A process which is predictable - "apply stimulus - measure - output" can be easily modified to accomodate additional parameters or changes to their limits. TS software changes on the other hand, frequently require complex algorithm development, module, intermodule and system level checkout. Program regression is a very real danger. Every effort, therefore should be made to develop a clear, complete and practical allocated baseline for a TS system software.

Once the allocated baseline is released and approved at PDR, work can begin building a product baseline. Figure 5.3-2 depicts a typical simulator software program architecture broken down into functional elements.

This figure identifies functionally separate software elements without consideration of interface relationships or intermodule dependencies. These dependencies exist, however, and must be fully defined in the Interface Design Description (IDD) which must be made available at PDR and maintained under Class II control until FCA/PCA. These functions may be managed as one large CPCI or as several (four or more in the example of Figure 5.3-2). Regardless of level of CPCI designation, each major functional element or module design should be governed by individual -

a. Design requirements

b. Design description

c. External and internal interface descriptions

d. Flow diagrams

e. Narrative description

f. Test requirements

The CPDP should describe the type of documentation required to define the above information for each module. The foregoing design definition effort continues until CDR, at which time coding and debug begins. At this point the module designs, defined in released engineering documentation, are submitted to SCM for contractor internal control within the CPL. No changes can be made to these designs without authorization from the software design manager. This proviso insures that an approved design will not be altered by programmers subsequently coding to that design. Figure 5.3-3 depicts the levels of configuration control required from PDR (allocated baseline) through FCA/PCA (product baseline). It is seen that both the modular designs and the code evolve through increasingly tighter levels of configuration control. Concurrent with code and debug, the programmer conducts module tests until he is satisfied that his code functions in accordance with the module performance requirements. He then submits his code to the CPL where it is "formally configured" for the first time. It is given a configuration number (see paragraph 4.3) and stored for integration with other modules to form the next higher module or CPCI version. When all functional modules are available for a given CPCI version, they are integrated (assembled, compiled, and linked under support software control) to form the first CPCI "version". This "version" is then subjected to intermodule compatibility tests, modified as required under SCM controlled conditions (further discussed in paragraph 7.3), and then turned over for formal QA acceptance testing. At this time no changes can be made to the source code of any module without a contractor committed Class II change. This proviso insures all design subgroups potentially impacted by the change have an opportunity to evaluate impact. Finally after QA acceptance (to approved contractor internal validation procedures), FCA/PCA is held to approve the product baseline at which time Class I (customer approval) control prevails.
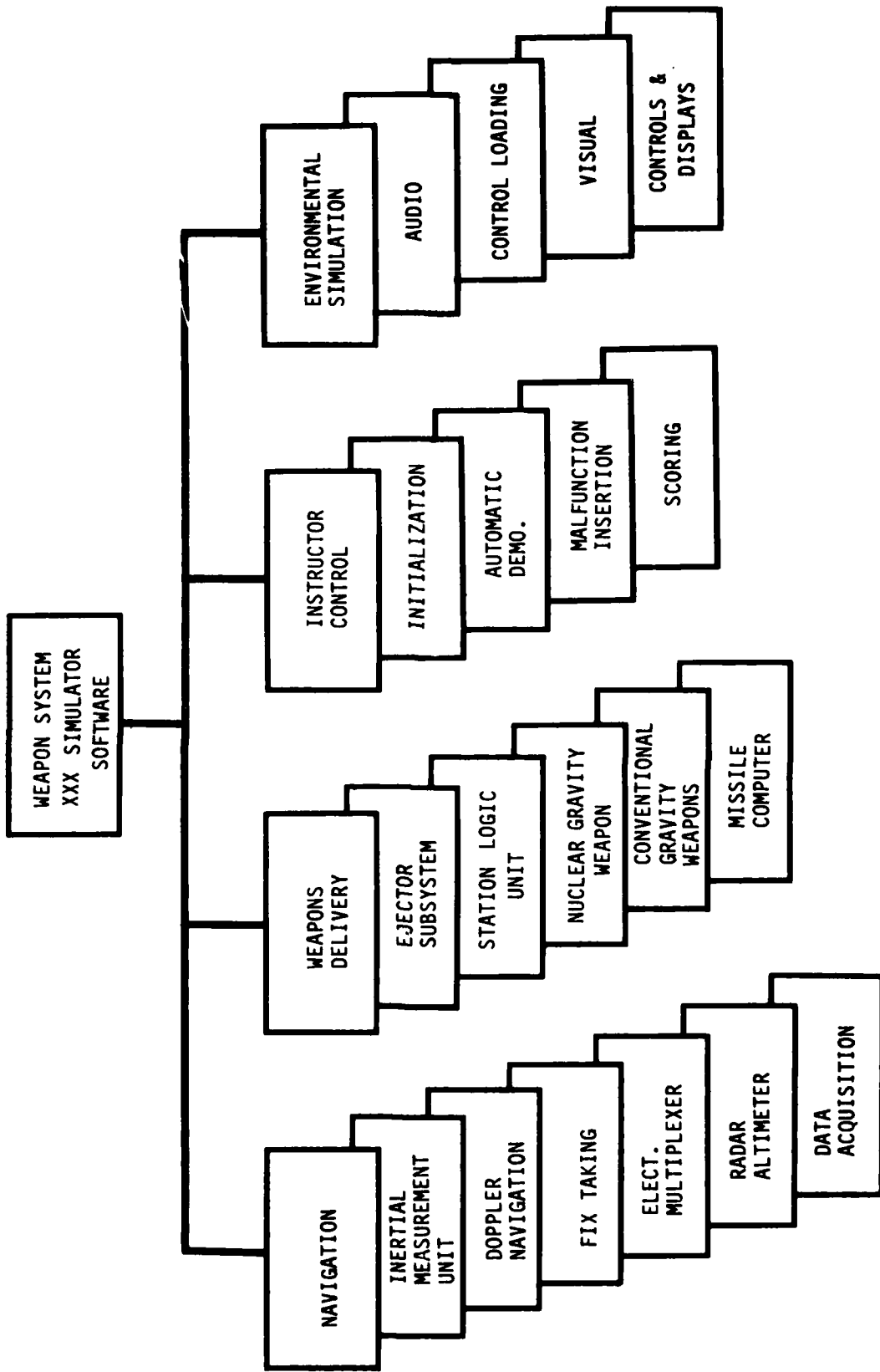
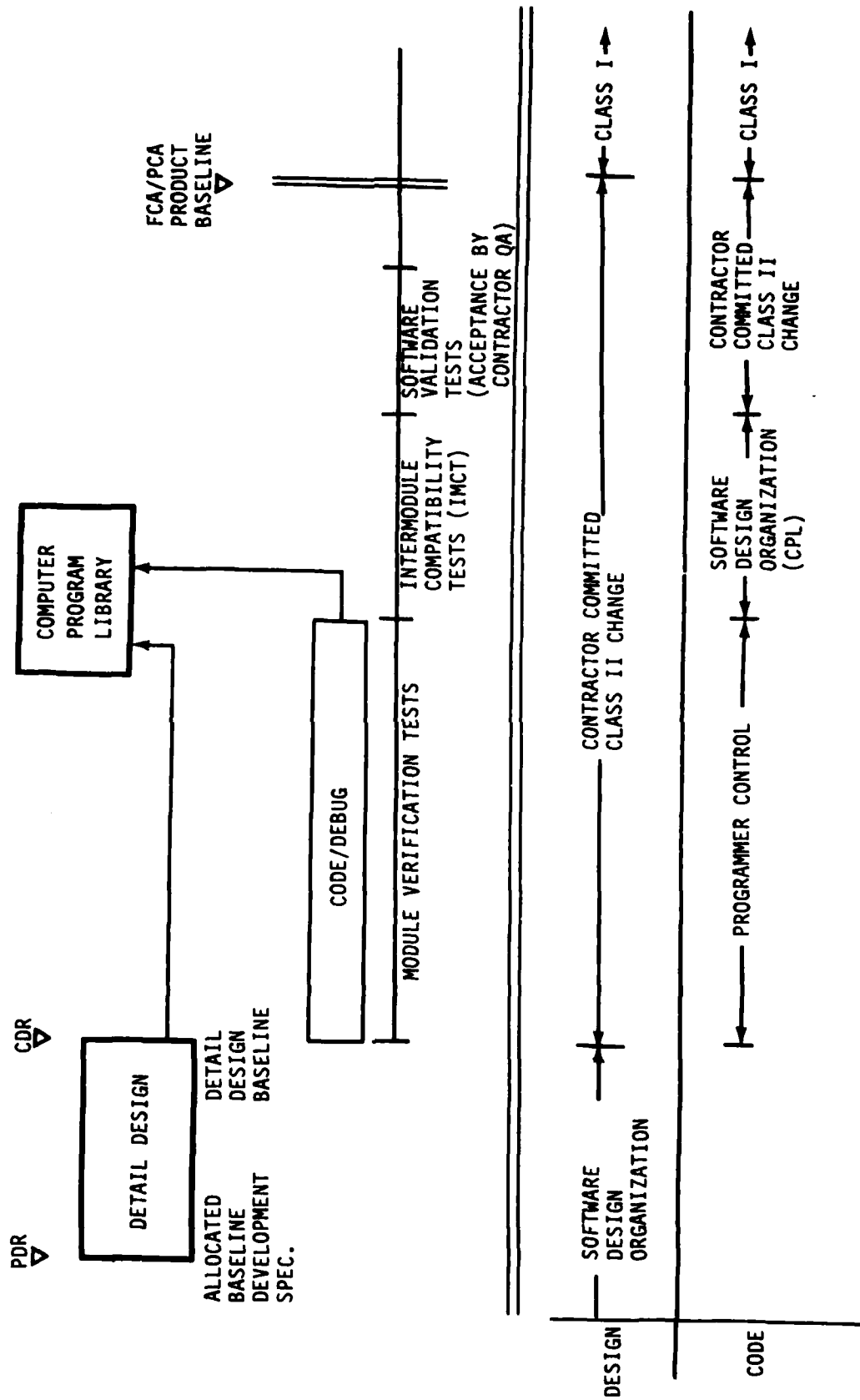35

Figure 5.3-2. TS Software Functional Elements

Figure 5.3-3. Configuration Control Phasing

37

5.3.2.1 "Block Change" Approach. Frequently changes originating from the various design subgroups are so voluminous that configuration control becomes difficult and unwieldy. Literally hundreds of "make work" type code modifications may be accumulated over several months against a given CPCI version. Such changes can be incorporated in two ways.

a. Handload or "patch" the object program

b. Recompile the modified source code

There are undesirable aspects of using either method exclusively. "Patching" the object program with hundreds of changes causes program performance regression. Recompiling a new total program to incorporate every minor change and then running a complete retest, is costly and inefficient.

A compromise is achieved by using the "Block change" approach. In this method changes are packaged together or "blocked" in accordance with some ground rule for manageability. Factors determining block size might be

a. Quality of changes

b. Schedule - critical project milestones

c. Major redesign

d. Merge of two or more CPCIs into one

e. Other technical/management reasons

Under this system, all changes accumulated under "Block 1" are identified, tracked and controlled by SCM. At a con-

venient point, patching of the Block 1 CPCI versions is terminated, the source program is updated to incorporate the intent of all Block 1 changes, and a new "Block 2" version is compiled and completely retested. Each successively developed "Block" may require several CPCI versions to accommodate a "test-debug-patch reconfigure-retest" cycle. This block approach is depicted schematically in Figure 5.3-4. Under this approach the first version of new "block" is a new recompilation of the original or updated source program. It is retested in total - to verify all performance requirements are met. As the design matures, changes are required to accommodate performance improvements or to fix errors. Each individual change or error is evaluated and a "patch" is designed to the object program. The code is changed and that portion of the test requirements affected by the patch is retested. The patch is logged and eventually a corresponding source code change is made and "rolled in" at the next "Block".

Under this approach, software performance is evaluated on a system level early in the development cycle to uncover major system level design incompatibilites. Each version performs slightly better than the previous. Each "Block" performs better than the previous Block, until all Part 1 requirements are met or exceeded.

5.3.2.2 Baseline Control Mechanisms - Summary. The preceding discussion describes the "concepts" of baseline control and the variants in control concepts required to accommodate inherent differences in ATE and TS software. The following section discusses methods of managing the mechanizing change control.
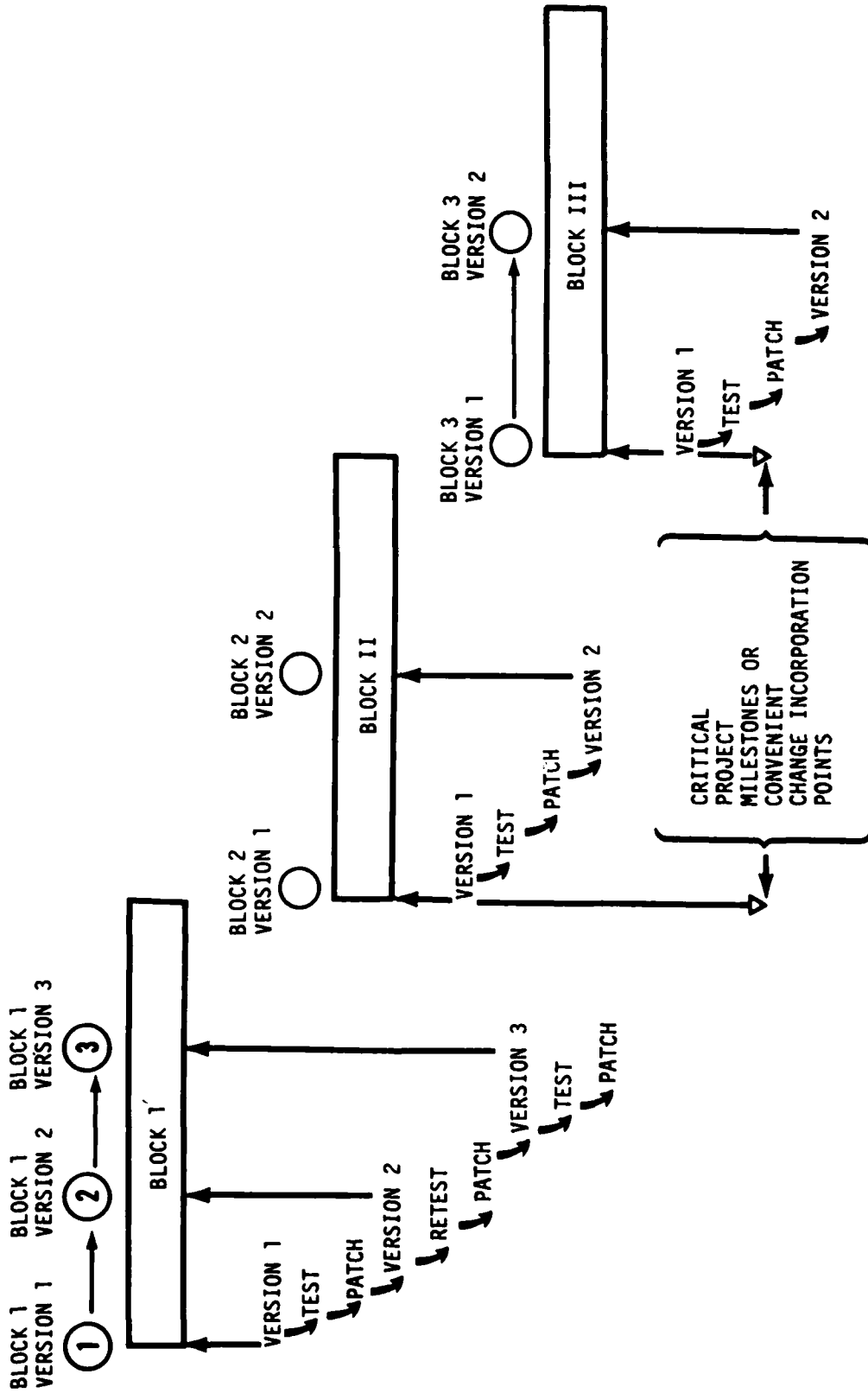
Figure 5.3-4. Block Change Approach to Change Control

Section 6.0   CHANGE MANAGEMENT

The subject of software change management embodies the preparation, evaluation, approval, incorporation, accounting, and test of all customer and contractor initiated changes to CPCI's. Change management encompasses changes in requirements, hardware and software interfaces, problem and errors, enhancements, etc. It includes change board activities, change reporting and statusing, problem reporting, tracking and corrective action. Some of the unique problems of change management for ATE and TS software are discussed in this section.

Change management embodies the following elements

   a. Change classification

   b. Change approval and release

   c. Change accountability and verification

The following subsections discuss each of these elements as they apply to ATE and TS software.

6.1   CHANGE CLASSIFICATION

Formal changes to CPCI's are classified as Class I (design) or Class II (discrepancy) in accordance with MIL-STD-483, Appendix XIV. An engineering change is classified as Class I when it affects the contractually specified form, fit, function, cost or delivery schedule of a CPCI. An engineering change is classified as Class II when it does not fall within the criteria for Class I. Once an external baseline is established (see Figure 5.1-1), all changes are processed as Class I. Every request for change is evaluated by configuration management to determine classification and type. The criteria for classification is as follows:

6.1.1   Class I Change Criteria

All proposed engineering changes to CPCI's are designated as Class I changes if one or more of the following are affected:

   a. Contractual specifications (functional, allocated and product baselines)

   b. Contractual plans

   c. Cost, fees, incentives

   d. Key milestone schedules, deliveries

   e. Statement of work (SOW) and/or the contract terms and conditions

   f. Government furnished equipment (GFE), safety, delivered manuals

6.1.2   Class II Change Criteria

All changes not considered to be Class I are designated as Class II when they are encountered during a period designated as "contractor committed Class II change control."

6.1.3   Change Processing

Formal change processing requires that the problem be identified, and processing initiated on the appropriate change paper. The proposed change request is classified as Class I or Class II and is coordinated with affected organizations to determine its scope and impact. It is then presented to the program change board for evaluation of the impact on cost, schedule, design, etc., and for approval or disapproval. The change paper and method of processing formal changes is dependent upon the classification of the change. This is described as follows:

6.1.3.1   Class I Changes. Class I changes are processed as either engineering change proposals (ECP's) or con-

41

tract change proposals (CCP's). They must be approved by the customer prior to implementation.

a. An ECP is used to propose an engineering change to a contractual specification or an approved configuration identification/baseline. The ECP is a comprehensive document which contains provisions for supplying all the information necessary to make a thorough evaluation of the change and its impact on the entire system.

b. A CCP is used to propose a non-engineering change to the contract requirements; e.g., changes to the SOW, contractual schedules, contractual plans, equipment quantities, costs, etc.

6.1.3.2 Class II Changes. Class II changes are processed as either a committed or noncommitted change. Class II committed changes are identified as committed changes and Class II non-committed changes are as noncommitted changes, or liaison changes. Class II committed changes may be implemented after the program change board approval and commitment is established, without prior customer approval; however, they must be submitted to the customer for concurrence with the classification. Class II noncommitted changes do not require change board action or commitment.

a. A committed change is any Class II change requiring interorganization coordination through the program change board for scheduling and release commitment. Committed changes are serially numbered.

b. Liaison (noncommitted Class II): This type of Class II change is made to permit software conformance to the intended design. It is fully coordinated by the design engineer with the liaison engineering, quality assurance and manufacturing planning organizations.

6.1.4 Software Change Initiation

During the software development process, design deficiencies or coding errors may be discovered and a change to the software (design documentation or code) may be required. These deficiencies or errors must be documented and reported on appropriate forms so that the problem may be analyzed and appropriate action taken.

The type of form used to report these problems and the type of change control required depends on the stage of software development, whether an external or internal baseline is affected and whether the deficiency or error is a design error or coding error.

Irrespective of the phase of development, all changes precipitated by software errors should be reported on a software problem report (SPR), (see Figure 6.1-1). This form provides for orderly definition, solution and tracking of all software errors. It is also the prime vehicle for analyzing trends in software discrepancies.

Design changes not due to errors (performance improvements, requirements changes) are described and authorized on a Design Change Request (DCR) or equivalent - (Figure 6.1-2). These forms are internally controlled by the Software Design and SCM organization. During formal change control periods (Figure 5.3-3), they do not in themselves authorize software drawing changes, but rather define the reason for and description of an actual software change. The engineering drawing or document is changed only by committed change. Changes originating from either errors or design change are evaluated and authorized by change board activity to be discussed next.

6.2 CHANGE APPROVAL AND RELEASE

In organizing for management of complex software controlled acquisition systems, a contractor may establish two change

SOFTWARE PROBLEM REPORT

Problem Report No. _____

Project Name _____ Computer/Lab Utilized _____ Program _____

_____

| Problem Discovery | Name of finder _____ Date _____ |

Method of detection:
- ☐ Usage
- ☐ Inspection/Analysis

- ☐ Development test
- ☐ Integration test
- ☐ Acceptance Test

Tools used to detect:
- ☐ None
- ☐ Design review
- ☐ Peer review

- ☐ Dump (terminal)
- ☐ Dump (dynamic)
- ☐ HOL Debug
- ☐ Analyzer

- ☐ Simulation
- ☐ Assertion
- ☐ Proof
- ☐ Other

Description of symptoms _____

_____

_____

Configuration level _____

Correction importance/need date _____

Authorizing Signature _____ Orgn. _____ Date _____

_____

| Problem Analysis | Name of analyst _____ Start date ____ End date _____ |

Findings _____

_____

Resources expended: person _____ computer _____

Estimated resources to correct: person _____ computer_____

_____

| Problem Correction | Name of programmer _____ Start date____ End date_____ |

Description of Correction _____

_____

_____

Components changed and configuration level _____

Resources expended: Person _____computer_____

Problem category –
- ☐ Job control language
- ☐ Operational Interface
- ☐ Coding error – data declaration
- ☐ Coding error – executable instruction

- ☐ Design error – omitted logic
- ☐ Design error – faulty logic
- ☐ Testing
- ☐ Configuration management

- ☐ Documentation
- ☐ Other

_____

Final Authorizing Signature _____ Orgn._____ Date _____

Figure 6.1-1. Software Problem Report

DESIGN CHANGE REQUEST (DCR)

Report No._____

Originator_____

Organization   Date
_____

Reason for Change

_____

Proposed Solution

_____

CI's/CPCI's Impacted   Documents Affected

    CI No.   Rev.   Version   ● IDD   / /
                                    ● Part I Spec   / /
                                    ● Part II Spec / /
                                    ● VDD   / /
                                    ● Operating & Inst./ /
                                    ● Test Procedures / /

Change Description

                                    Programmer_____
_____

Approvals

        ● Software Design_____

        ● Quality ASsurance_____

        ● Systems Eng. _____

        ● Hardware Design_____

        ● Test and Evaluation_____

_____

Figure 6.1-2. Design Change Request

44

board functions. One is the regular change board which process and commits all system changes, hardware and software. In addition, in order to manage the large volume of software development changes which determine the evolution of the ultimate operational program, a software change board may be established. Its function is to coordinate proposed changes in software design among programmers, hardware designers, interface specification managers and systems engineers to insure the compatibility, correctness and documentation of all program changes. This board is chaired by the SCM group which insures the coordinated approval, documentation, incorporation and test of all changes are controlled and recorded. Module level changes to CPCI's not yet committed to formal release are normally excluded, however, once released CPCI's should require this level of coordination.

6.2.1 Project Change Board

The project change board meets periodically to act on proposed Class I and Class II changes. This board ensures that changes are properly classified; that their effect on interfacing engineering disciplines, cost and schedules are properly assessed; and approves the implementation of Class II changes and the transmittal of proposed Class I changes to the customer for approval or disapproval. SCM is a member of this change board. SCM will support this change board to control proposed software changes, assure completeness, and assess impact on documentation.

6.2.2 Software Change Board

A software change board or software configuration control board is established and chaired by the software manager or his designee to provide a means of controlling and processing internal software changes. This board reviews all software group originated changes and approves or disapproves the change before it reaches the project change poard (or during periods of internal

control only). Board members are appointed by the software manager and normally include software design, test, quality assurance and systems engineering representatives. The software change board working in conjunction with the SCM organization is responsible for change approval and release. A typical flow for internal software changes is shown in Figure 6.2-1.

When change control progresses into formal Class II control phases (CDR for design; completion of Intermodule Compatibility Tests (IMCT) for coding, see Figure 5.3-3) the software change board (SCB) functions continue unchanged. However, the SCB approved change must then be committed and scheduled in the project change board. The project change board assures the necessary drawings and documents are changed and appropriate retesting are accomplished to implement the SCB submitted change. Once all changes are initiated, authorized and scheduled by the applicable change board they are incorporated as changes to the source and object documentation and media (decks, tapes, listings). This change incorporation and verification is called configuration accountability and is discussed in the following section.

6.3 CHANGE ACCOUNTABILITY AND VERIFICATION

Change accountability is one of the most essential tasks in achieving configuration integrity of a deliverable CPCI. Unless properly managed during development and validation testing, configuration control can be easily lost. This danger arises primarily because the program submitted for the start of the validation (or acceptance) testing is frequently significantly different from the program which completes testing. For example, in TS software, the version submitted for validation (system level tests) is normally the version completing verification (single thread,
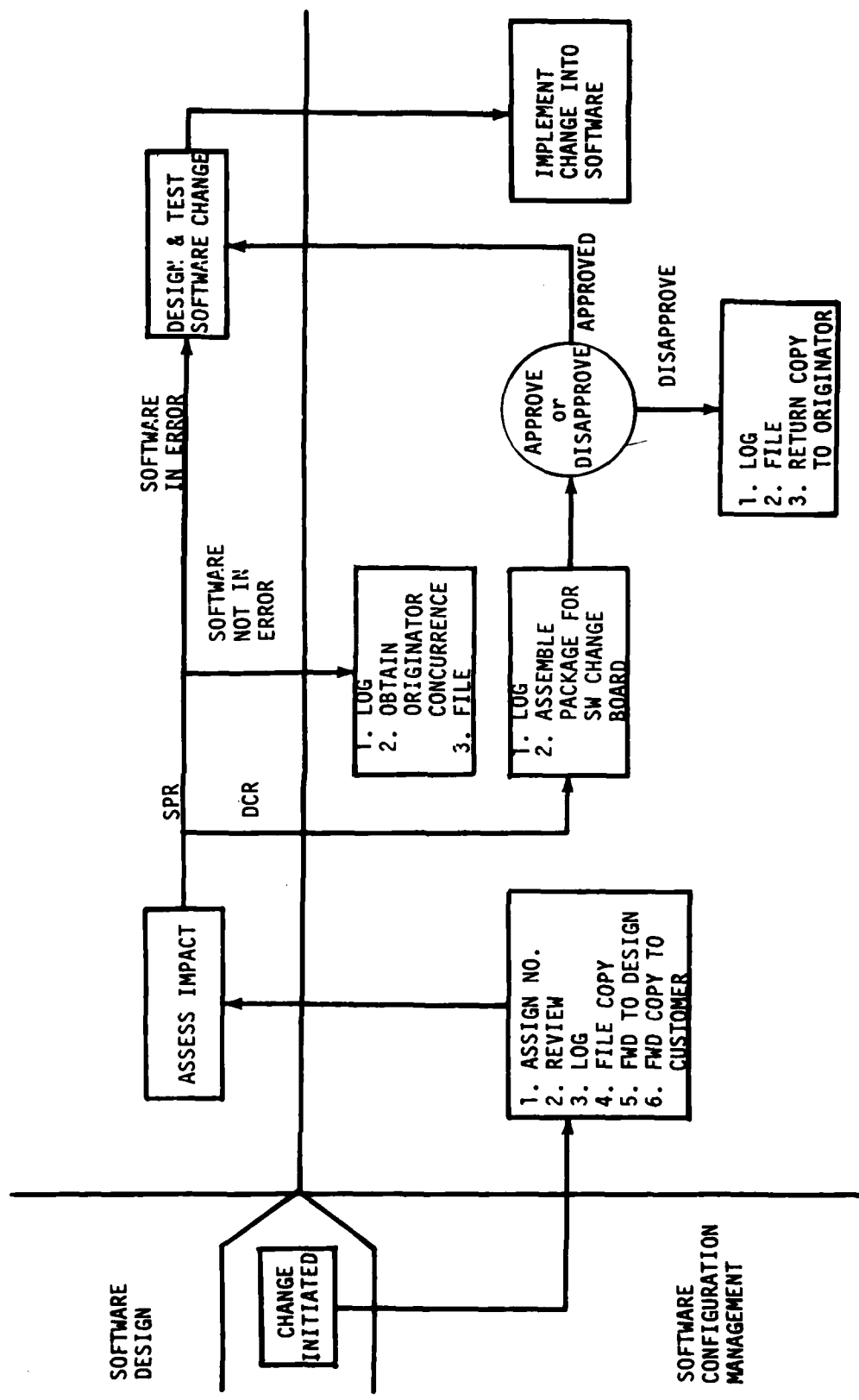
45

Figure 6.2-1. Internal Software Change Control

46

IMCT). It has never been formally executed in a system environment. Understandably numerous problems arise. As those problems increase in number throughout the validation test period (which may be several weeks or months) tremendous tracking and control problems can arise. For example:

a. Test conductors may change with shift change.

b. QA personnel/technicians change.

c. At what point in the test procedure was the problem encountered?

d. What was the fix? - patch? recompiled tape?

e. What locations and data were modified?

f. What controls insure only authorized memory locations were modified after initial load?

g. What retest was done --

(1) - for the portion of the program failing?

(2) - for possible regression of the program in other areas?

h. At what point in the procedure was official testing resumed?

i. Who authorized the fix and test restart?

j. Has the impacted released engineering been changed?

Unless rigorous control is exercised by QA personnel, any of the above factors can create a configuration loss. It would be ideal to run all validation tests so that official "sell-off" would be a smooth formality for QA and the customer. Unfortunately schedules don't always allow this. The result, if uncontrolled testing prevails, is software which is inadequately tested or questionably configured. We shall examine methods of achieving the required accountability.

Configuration accountability during the pre-formal test phases is accomplished by the programmer through his notes and engineering master tapes and listings. When modules are functionally complete (module verification test (MVT) complete), the source program listing (or symbolic deck) is submitted to the CPL for control by SCM personnel. This deck is the source program for that module; together with the narrative description and flow diagram its configuration definition is complete up to that point. Between the completion of MVT and start of software validation tests (SVT), CPL personnel must account for the configuration of that module and all like modules. Upon completion of Intermodule Compatibility Tests (IMCT), the primary source of further changes is SVT. This is the formal acceptance test of the software for contractor QA personnel. (See "Software Quality Assurance" guidebook, section 6 for a discussion of formal tests.) Incorporation of "hand patched" code changes are physically verified by QA via one of the following techniques.

a. Use of card reader

b. Use of key board with CRT

c. Use of interactive text editor

d. Use of data cassettes

In each of the above mechanisms, QA can verify configuration by observing data changes entering memory at designated addresses. Upon completion of testing the final configuration of the program is the original program tape plus a verified card deck representing drawing authorized handpatches. Alternately, the original load tape, modified by patches and resident in memory, may be dumped on to another blank tape to configure the accepted program.

Under this mode of configuration accountability, a previously validated (acceptance tested) version of a CPCI could be "patched" and only require verification of the changed addresses and interfaces. When a newly recompiled version is submitted for validation, however, it will most likely entail a complete retest since the effect of recompilation, assembly edit, etc. on the revised source code demands a complete re-evaluation of the new object program. New machine code is no longer physically comparable to previously validated code, and QA must regard the new program as an unvalidated baseline or starting point for controlling a new set of handpatches.

For ATE programs, many of the automated configuration control aids described in paragraph 5.3.1 handle the task of accounting for program code and in some cases flow diagram changes. When used in conjunction with authorized procedures, their use simplifies and streamlines program configuration accountability.

Configuration accountability however, encompasses more than just code accountability. Configuration status accounting documentation is the means through which actions affecting CPCI's are recorded and reported to program and functional managers. It principally records the "approved configuration baseline" and the implementation status of changes to the baseline. In this way, it provides managers with confirmation that change decisions are being implemented as directed. Configuration status accounting is the recording and

reporting of the information that is needed to manage configuration effectively. It includes listing the approved configuration identification, the status of proposed changes to configurations, and the implementation status of approved changes. It involves maintaining and reporting the status of CPCI specifications, associated documents and proposed changes. The system utilizes two primary reports; the Configuration Index and the Change Status Report.

The computer program configuration index provides the official listing of the CPCI specifications, drawings and other significant support documents. It identifies all approved changes and shows the current status of all CPCI documentation such as the computer program development and product specifications, test plans/procedures/reports, handbooks, manuals and the VDD. The change status report lists all proposed changes to the CPCI documentation listed in the configuration index. It provides information on the current status of the CPCI and changes throughout its development. QA uses the configuration index and change status report for change accountability to assure that all Class I changes incorporated into the software have been approved by the customer and that no unapproved Class I changes are incorporated.

The guidebook on "Software Quality Assurance," Section 6.0, describes some of the contractor internal procedures used to achieve software configuration accountability through the manufacturing record system.

Section 7.0   COMPUTER PROGRAM LIBRARY

The Computer Program Library (CPL) function is a specific requirement of MIL-S-52779, Software Quality Assurance Program Requirements and is treated in depth in the guidebook on Software Quality Assurance. However, since it is a vital tool in maintaining software configuration control, its function is briefly summarized herein.

7.1   SOURCE AND OBJECT CODE CONTROL

All source materials used to configure a CPCI should be placed under internal control after MVT. This is the point at which the programmer has declared his module to be satisfactory for integration with other modules. It should be submitted along with appropriate decks, listings, flow diagrams, descriptions, notes, in a standard format for integration with other like modules. Submission of code to the CPL is accompanied by the assignment of formal software component part numbers - module or program segment numbers which can be called up on an engineering configuration drawing such as the VDD. It is ready now for the first phase of formal control. Unless changed for authorized reasons, it is ready for compilation and assembly into an object program.

7.2   MEDIA AND DOCUMENTATION SECURITY

The CPL must be a secure repository which protects configured media and documentation from unauthorized modification, loss or damage. Just as released engineering drawing masters are protected in a secure vault, programming records require similar security. It is costly to reproduce an object code program if during formal testing the test copy tape, disk etc. is accidentally damaged or lost. If the source materials used to build the program master tape, disk, etc. are lost, results could be catastrophic.

Equipment and facilities specifically designed for organized and secure storage of computer media and documentation are available commercially. These facilities include:

a. Storage cabinets for tapes, disks, cassettes

b. Environmentally designed vaults

c. Mobile cabinetry for listings, operating manuals, etc.

d. Color coded tape seal rings for tape categorization

e. Tape disk labels

f. Cypher locked cabinets for top security items

g. Certified magnetic media

This available equipment can be purchased by modular components and arranged by simple remove and replace operations to reconfigure storage facilities to meet changing needs. A good CPL should utilize equipment of this sort and be adequately documented to describe location and access requirements for users.

7.3   SUBMISSION OF SOFTWARE FOR TEST

As the programmers submit completed modules to the CPL for integration into a CPCI, they will likely be in various degrees of legibility, completeness, conformance to prescribed standards and conventions for programming and flow charting. It is the responsibility of SCM to assure either within the library function or prior to submission, that these components meet prescribed format requirements. The library then is responsible for tracking and assuring incorporation of all approved changes to these modules or components and correlating them to requirements for the various phases of testing. For example, it is probable that various programs,

49

representing progressively maturing versions of the same CPCI, may be developed in parallel. In other words, Version 2 of CPCI X may be just beginning test while Version 1 of the same CPCI is completing test. Changes applicable to each version must be distinguished if changes have different version effectivities.

It is a CPL responsibility to assure that all materials needed to assemble the deliverable media are properly stored and maintained. It is an SCM responsibility to assure that those materials are properly configured by released engineering. The integration of responsibilities for defining, building and verifying deliverable media is shown in Figure 7.3-1.

Building Materials for Deliverable Software	Responsibilities		
	SCM	CPL	QA
• Source Listings • Symbolic decks • Compilers • Assemblers • Link Editors • Loaders • Code Generators • JCL • Host Utilities • Data bases • Physical media • Object listings	• Determine required materials • Release Configuration Descriptions	• Identification • Storage • Maintenance	• Verification • Audit CPL controls • Accept deliverable media

Figure 7.3-1. Integration of Media Management Responsibilities

Section 8.0 REVIEWS AND AUDITS

Formal reviews and audits are the project milestones which formally demarcate changes in development phases and provide a basis for product acceptance by the customer. These reviews and audits are essential to the efficient development of quality software. Design reviews assess the completeness and evaluate the results of major development phases before proceeding with the next phase. Formal audits are conducted after the software system testing has been completed to determine whether the software and supporting documentation meet contract and project requirements prior to system acceptance. The guidebook on "Reviews and Audit" discusses this subject in depth. It is discussed synoptically here since it is an integral part of configuration management.

8.1 DESIGN REVIEWS

8.1.1 System Requirements Review (SRR)

The SRR will be conducted at the conclusion of the system analysis phase to establish the adequacy of the definition of the system technical requirements. The entire system engineering response to the SOW and specification requirements will be reviewed. The software/hardware trade studies will be reviewed to determine that they are sufficiently comprehensive and complete for allocating functions to software. The allocated software functions will be uniquely defined and not be dependent on other requirements. Performance limits will be designated in units of measure appropriate to software and will be verifiable. This review will cover the external hardware interfaces and dependencies.

8.1.2 System Design Reviews (SDR)

SDR's will be held at the point when system characteristics have been defined and functions have been allocated to configuration items, both CI's and CPCI's.

These reviews will evaluate the optimization, correlation, completeness, and risks associated with the defined allocation of requirements. In addition, the engineering process that produced the allocation will be reviewed.

Software requirements will be reviewed for technical adequacy, completeness, and clarity. Software verification requirements will be reviewed for completeness, adequacy of methods, and traceability and compatibility with higher tier specification requirements. All applicable software standards for design and programming will be identified. The software test plan will be reviewed to ensure that it satisfies the test requirements of the development specifications.

8.1.3 Preliminary Design Review (PDR)

The PDR will be conducted at the conclusion of the preliminary design phase to evaluate progress and the technical adequacy of the basis design approach prior to the detail design effort. A PDR will be conducted for each CPCI.

This review will confirm that the initial portion of the computer program product specification incorporates and satisfies all requirements in the computer program development specification. The allocation of CPCI requirements to individual modules, the allocation of storage to computer programs, timing estimates, sequencing requirements, operational concepts, and data base structure and organization will be reviewed. All functional interfaces between CPCI's and hardware CI's will be reviewed for consistency and compatibility with interface control drawings and basic design specification.

8.1.4 Critical Design Review (CDR)

The CDR will be conducted at the conclusion of the detail design phase for each CPCI to ensure that the detail design solutions satisfy the performance

53

and engineering requirements of the computer program development specifications. This review will establish the integrity of the computer program logical design prior to coding and testing. The review will establish the compatibility of the module design descriptions with the basic design description and the data base design description. All external and internal interfaces will be reviewed to establish system compatibility of design.

The software test plan will be reviewed to ensure that it reflects the current information developed during detail design. All computer program test procedures will be reviewed for compatibility with the design and test requirements and for adequacy. The status of all changes will be reviewed to ensure that all approved changes have been incorporated in affected documentation and that proposed changes have been initiated with appropriate change paper.

8.2   CONFIGURATION AUDITS

8.2.1   Functional Configuration Audit (FCA)

The FCA is a formal examination of the functional characteristics and test data for each CPCI prior to acceptance. The FCA will verify that the CPCI's actual performance complies with the performance requirements of the computer program development specification. The test plans and procedures and the test data accumulated during software testing will be reviewed to verify that the item has performed as required. The results of analyses or simulations for those requirements that cannot be verified by test will be reviewed to ensure their validity. All ECP's and PDR/CDR

initiated changes will be reviewed to ensure that they have been incorporated and verified. A list of all documentation of the CPCI will be reviewed to ensure adequate documentation of the physical configuration for which test data has been verified. All applicable computer program manuals (user's, programmer's, and operator's) will be reviewed. The current draft of the computer program product specification will be available for examination to provide guidance for conduct of the PCA.

8.2.2   Physical Configuration Audit (PCA)

The PCA is a formal examination of the "as-built" configuration of each CPCI prior to acceptance and delivery. It comprises an examination of the CPCI against its technical documentation to establish the CPCI's product configuration identification. The PCA will evaluate the adequacy of the acceptance testing requirements, audit the engineering drawings, specifications, technical data, tests, technical descriptions, flow charts, listings, and operating manuals. The VDD will be examined to ensure completeness of descriptions and records of change control.

The completion of FCA/PCA constitutes a product baseline. All changes in design or product configuration must be processed as class I (customer approval). In most cases FCA/PCA is prerequisite to product acceptance by the customer. Ultimate final acceptance is contingent upon delivery installation and operational certification by the customer.

Section 9.0 BIBLIOGRAPHY

1. DOD Weapons System Software Management Study, May 1975, John Hopkins Univ., Applied Physics Lab.

2. DOD Directive 5010.21 Config. Mgmt. Implementation Guidance, August 1968.

3. Industry/Joint Services Conference/Workshop - ATE - 3-7 April 78 San Diego.

4. McGowan, Kelly - Top Down Structured Programming Petrocelli/charter, 1st edition 1975.

5. Structured Programming in a Production Programming Environment, F. Terry Baker, IEEE transactions on Software Engineering, Volume SE-1 No. 2 June 1975.

6. H. P. Atlas compiler Operating Manual 92100-93001 - Hewlett Packard Co.,/Automatic Measurement Division, Sunnyvale, CA 1976.

7. ATE software configuration management, Dennis L. Wood, President Software Enterprises Corporation, Canoga Park, Calif. ASSC record 72, p. 110.

8. D180-19846-1 Software Quality Assurance Guide Boeing Company Rev. B, 23 December 1977.

Section 10.0   MATRIX: GUIDEBOOK TOPICS VS. GOVERNMENT DOCUMENTATION

The elements in the attached matrix, fig-
ure 10.0-1 contain Government document
sections   concerned   with   principal
guidebook topics.

TOPICS \ DOCUMENTS	MIL-S-xxxxx/xxx	SDS EXHIBIT 61.xxx	AFR xxx-14 VOL.II	DODI 5xxx.xx	DODD 5xxx.xx	DOD-TR-xx-xxx	AFR 65-3	MIL-STD-483	MIL-STD-490	MIL-STD-499	MIL-STD-1521	MIL-STD-xxx	AFR-xx-1	AFR-xxx-1	AFSCP xxx-1	MIL-S-xxxx	MIL-STD-xxxx
BASELINE MANAGEMENT			CH 6	SEC V INCL 2				SEC 3									
CDR	3.2.6		CH 6								SEC 3 APP D						
CHANGE CONTROL			CH 6	SEC VI				APP XIV									
CHANGE STATUS LIST								APP VIII									
CMP			CH 7					SEC 3 APP I									
COMP.PROG.DEV.PHASES			CH 2		SEC V												
COMP.PROG.DEV.SPEC.						SEC III		APP VI	APP VI								
COMP.PROG.INTERFACES			CH 6					APP II									
COMP.PROG.MANUAL						SEC II											
COMP.PROG.PRODUCT SPEC.						SEC III		APP VI	APP XIII								
COMP.PROG.USERS GUIDE						SEC III											
CONFIG.INDEX								APP VIII									
CONFIG.MANAGEMENT	3.2.8		CH 6	●	SEC V		●	●									
CONTRACTOR DOCUMENTS		●	CH 7			SEC III											
CPCI					SEC V			SEC 3									
CPUP			CH 3 CH 7					SEC III									
LIBRARY CONTROLS	3.2.5																
SOFTWARE STANDARDS	3.2.6																
SOFTWARE DOCUMENTATION	3.2.7	●															
CONFIGURATION STATUS ACCOUNTING												●					
ECP								APP XIV									
PCA			CH 6	SEC VIII				SEC 3 APP XII			SEC 3 APP E						
LIFE CYCLE PHASES			CH 2			SEC III									ATT 3	CH 1	
PCA			CH 6	SEC VIII				SEC 3 APP XII			SEC 3 APP F						
PDR			CH 6								SEC 3 APP C						
RFP			CH 7														
SCN								APP VIII					●				
SDN			CH 7 CH 8													●	
TRD																	●
TS SYS SPEC.																●	
VDD						SEC III		APP VIII									

Section 11.0   GLOSSARY OF TERMS

Allocated Baseline - The approved configuration item identification. It governs the development of selected configuration items that are part of a higher level specification, e.g., system specification. It is usually defined by the Computer Program Development Specification.

Acquisition Engineer - Military or civilian member of a SPO or an AFSC division who supports the activities of a SPO.

Baseline - An authorized documented technical description specifying an end item's functional and physical characteristics. It serves as the basis for configuration control and status accounting. It establishes an approved well-defined point of departure for control of future changes to system or equipment.

Certification - The test and evaluation of the complete computer program aimed at ensuring operational effectiveness and suitability with respect to mission requirements under operating conditions.

Computer Program Configuration Item - A computer program or aggregate or related computer programs designated for configuration management. A CPCI may be a punched deck of cards, paper or magnetic tape or other media containing a sequence of instructions and data in a form suitable for insertion in a digital computer.

Configuration Management - A discipline applying technical and administrative direction and surveillance to:

    a. Identify and document the functional and physical characteristics of a configuration item

    b. Control changes to those characteristics; and

    c. Record and report change processing and implementation status

Control Software - Software used during execution of a test program which controls the nontesting operations of the ATE. This software is used to execute a test procedure but does not contain any of the stimuli or measurement parameters used in testing a unit under test. Where test software and control software are combined in one inseparable program, that program will be treated as test software (AFLC 66-37).

Data Base - A collection of program code, tables, constants, interface elements and other data essential to the operation of a computer program or software subsystem.

High Order Language - Problem or system oriented code which can be automatically translated to machine language either directly or indirectly (through an assembly language step).

Host Computer - An off-line, general purpose, programmable computer which prepares data or code for a (target) system computer, e.g.: ATE central computer.

Product Baseline - The final approved configuration identification. It identifies the as designed and functionally tested computer program configuration. It is defined by the Computer Program Product Specification at FCA/PCA.

Software Quality Assurance - A planned and systematic pattern of all software - related actions necessary to provide adequate confidence that computer program configuration items or products conform to establish software technical requirements and that they achieve satisfactory performance.

Software - A combination of computer programs, documentation and computer data required to enable the computer equipment to perform computational or control functions and to enable program maintenance.

59

Software Maintenance - Any change to previously established software. Sources for such change are coding errors, module design problems, system interface problems, revised system requirements, and capability improvements.

Support Software - Auxiliary software used to aid in preparing, analyzing and maintaining other software. Support software is never used during the execution of a test program on a tester, although it may be resident either on-line or off-line. Included are assemblies, compilers, translators, loaders, design aids, test aids, etc. (AFLC 66-37).

System Life Cycle - The system acquisition life cycle consists of the following five major phases with major decision points:

a. Conceptual phase

b. Validation phase

c. Full-scale development phase

d. Production phase

e. Deployment phase

(AFR-800-14, Volume II)

Test Software - Programs which implement documented test requirements. There is a separate test program written for each distinct configuration of unit under test (AFLC 66-37).

Top Down Structured Programs - Structured programs with the additional characteristics of the source code being logically, but not necessarily physically, segmented in a hierarchial manner and only dependent on code already written. Control of execution between segments is restricted to transfer between vertically adjacent hierarchial segments.

Validation - System/software validation is the integration and test of all hardware and software components to assure the complete system fulfills all system requirements. Validation is generally regarded as the act of exercising the software after verification testing in a real or simulated operating environment against a precisely defined set of mission or test case requirements deemed representative of the usage environment for that product. Both verification and validation are required for software acceptance.

Verification - Computer Program Verification is the iterative process of continuously determining whether the product of each step of the computer program acquisition process fulfills all the requirements levied by the previous step.

Section 12.0 ABBREVIATIONS AND ACRONYMS

A/I - Adapter/Interface

ATE - Automatic Test Equipment

ATLAS - Abbreviated Test Language for
All Systems

CCP - Contract Change Proposal

CDR - Critical Design Review

CI - Configuration Item

CMP - Configuration Management Plan

CPCI - Computer Program Configuration
Item

CPDP - Computer Program Development Plan

CPL - Computer Program Library

CPPS - Computer Program Product Spec.

CPU - Central Processor Unit

CRT - Cathode Ray Tube

DCR - Design Change Request

DID - Data Item Description

ECP - Engineering Change Proposal

FCA - Functional Configuration Audit

GFE - Government Furnished Equipment

GFP - Government Furnished Property

HOL - Higher Order Language

IDD - Interface Design Description

IMCT - Intermodule Compatibility Test

ITA - Interface Test Adapter

LRU - Line Replaceable Unit

MVT - Module Verification Test

PCA - Physical Configuration Audit

PDR - Preliminary Design Review

PROM - Programmable Read Only Memory

QA - Quality Assurance

RFP - Request for Proposal

SAE - Software Acquisition Engineering

SCB - Software Change Board

SCM - Software Configuration Management

SDR - System Design Review

SOW - Statement of Work

SPO - System Program Office

SPR - Software Problem Report

SRR - System Requirement Review

SVT - System Validation Test

TPS - Test Program Set

TRD - Test Requirements Document

TS - Training Simulator

UUT - Unit Under Test

VDD - Version Description Document

Section 13.0 SUBJECT INDEX