(18) (19) ASD-TR-78-47

(2) NW

(14) D180-24700-1

(6)

SOFTWARE QUALITY ASSURANCE, One of the Software Acquisition Engineering Guidebook Series.

DIRECTORATE OF EQUIPMENT ENGINEERING
DEPUTY FOR ENGINEERING

(10) M.P. Kress

(12) 75

(11) JANUARY 1979

(9) TECHNICAL REPORT ASD-TR-78-47
Final Report.

(16) 2238

Approved for public release; distribution unlimited.

(15) F33657-76-C-0723

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

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Richard W. Ittelson

RICHARD W. ITTELSON,
Technical Advisor
Directorate of Equipment Engineering

Richard J. Sylvester

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

FOR THE COMMANDER

John S. Kubin

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

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify _____, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AIR FORCE/56780/17 March 1980 — 400

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ASD-TR-78-47	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE QUALITY ASSURANCE, Software Acquisition Engineering Guidebook Series		5. TYPE OF REPORT & PERIOD COVERED Final 6. PERFORMING ORG. REPORT NUMBER D180-24700-1
7. AUTHOR(s) M.P. Kress		8. CONTRACT OR GRANT NUMBER(s) F33657-76-C-0723 new
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 80
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, Software Quality Assurance, Software Quality Control, Software Configuration Control, Configuration Management, Hierarchial Decomposition, Validation, Verification, Analysis, Code and Checkout, Test

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 describes acquisition engineering and contractor tasks associated with quality assurance and control of software. It provides a practical guide to controlling software life cycle costs through quality assurance.

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

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

JRB

09610

FOREWORD

This guidebook was prepared as part of the Software Acquisition Engineering Guidebooks Contract F33657-76-C-0723 CDRLitem A00A. It provides the Air Force project office with software quality assurance guidelines for automatic test equipment and training simulator system acquisitions. It is written to provide planning guidance to the AF Project Office in establishing software QA requirements and implement guidance to local AFPR offices charged with contractor surveillance. It reflects the latest DOD published policy on software quality assurance while offering a practical and cost effective approach to total life cycle software quality assurance.

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.

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.

ASD-TR-78-43,	Computer Program Maintenance
ASD-TR-78-44,	Software Cost Measuring and Reporting
ASD-TR-78-45,	Requirements Specification
ASD-TR-78-46,	Computer Program Documentation Requirements
ASD-TR-78-47,	Software Quality Assurance
ASD-TR-78-48,	Software Configuration Management
ASD-TR-78-49,	Measuring and Reporting Software Status
ASD-TR-78-50,	Contracting for Software Acquisition
ASD-TR-79-5042,	Statements of Work (SOW) and Requests for Proposal (RFP)
ASD-TR-79-5043,	Reviews and Audits
ASD-TR-79-5044,	Verification, Validation and Certification
ASD-TR-79-5045,	Microprocessors and Firmware
ASD-TR-79-5046,	Software Development and Maintenance Facilities
ASD-TR-79-5047,	Software Systems Engineering
ASD-TR-79-5048,	Software Engineering (SAE) Guidebooks Application and Use

TABLE OF CONTENTS

TABLE OF CONTENTS - Continued

TABLE OF CONTENTS - Continued

LIST OF FIGURES

Section 1.0 INTRODUCTION

AF Policy (AF-74-1) defines Quality Assurance (QA) as a discipline which provides "adequate assurance that material, data, supplies and services conform to established technical requirements and achieve satisfactory results." This definition transcends earlier interpretations which assumed that QA should merely verify conformance to specifications. Software Quality Assurance (SWQA) concepts today, which are truly cost effective in the long term, reflect a total life cycle approach. Accordingly, methods presented in this guidebook are designed to achieve not only quality of conformance, but quality of specification and quality of design as well. True quality software satisfies a customer's needs, not simply a specification. SWQA, therefore, is an organizationally interdependent set of documented controls spanning all phases of the software life cycle. These controls are designed to minimize the quantity of post-delivery software deficiencies traditionally so prevalent in software systems. SWQA is not limited to the activities of a single contractor functional organization, nor to activities associated strictly with program acceptance. Rather, SWQA is a multiorganizational effort administered under the auspices of QA which spans all phases of system acquisition, and covers all activities which assures development and delivery of high quality software. Quality software is characterized by:

a. Complete, unambiguous requirements specifications.

b. Readily verifiable design.

c. Ease of program change and maintenance.

d. Documentation compliant with programming conventions and standards.

e. Accurate and complete records of requirements satisfaction.

f. Minimal post-delivery errors.

The Department of Defense, as reflected in numerous publications (see Section 2.0) nas recognized the necessity of SWQA, as vital to the on-time, within-target cost delivery of high quality software. This guidebook presents methods of implementation compliant with that policy.

1.1 PURPOSE

It is the specific purpose of this guidebook to present SWQA methodology tailored to Automatic Test Equipment (ATE) and Training Simulator (TS) system acquisitions. The guidebook will be used by AF acquisition engineering personnel to conceive, tailor, specify and enforce SWQA requirements in Request for Proposal (RFP's) and other contractual documents and specifications. It is also useful in preparing checklists for proposal evaluations and contractor surveillance activities. It should be recognized that acquiring ATE or TS software is an integral part of a larger system acquisition. Accordingly, the guidebook considers all system requirements impacting SWQA including applicable hardware development requirements, configuration management, data and design reviews, audits, and all modes of requirements satisfaction. The ultimate objective of these controls is to enforce disciplined Computer Program Configuration Item (CPCI) development controls and to provide official objective evidence of CPCI requirement satisfaction.

1.2 SCOPE

This Software Acquisition Engineering (SAE) guidebook is one in the guidebook

series related to ground systems, specifically ATE and TS. The guidebook titles are as follows:

Software Cost Measuring and Reporting
Requirements Specification
Contracting for Software Acquisition
Statement of Work (SOW) and Requests for Proposal (RFP)
Regulations, Specifications 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

The SWQA methods discussed in this guidebook encompass the following areas:

a. Estimating and planning the SWQA effort

b. Design review for verifiability and compliance with standards

c. Configuration verification and accounting

d. Verification and validation testing

e. Media identification and control

f. Use of software developmental and test tools

These controls implement AF regulations and specifications relating to computer program development and quality assurance. The guidebook describes how the SWQA function is documented, integrated into formal acquisition planning and implemented throughout all ATE and TS system acquisition phases. The guidebook is written for managers, engineers and quality assurance personnel with the Project Office (PO) responsible for ATE and TS software acquisition development, acceptance and operational deployment.

1.3 TS AND ATE OVERVIEW

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

1.3.1 Trainer Simulator System Characteristics

The TS system is a combination of specialized hardware, computing equipment and software designed to provide a synthetic flight and/or tactics environment in which aircrews learn, develop and improve the techniques associated with their individual tasks in an operational weapon system. In many cases, visual and/or motion systems may be included. Figure 1.3-1 depicts a typical training simulator system which employs digital processing capability.

The computation system, integral to the crew training simulator, consists of one or more general purpose computers. The computing hardware consists of machines with hardware floating point arithmetic and sufficient word size and memory to provide efficient use of the simulator High Order Language (HOL). When a multi-processor/ multi-computer system is used, it must be designed such that all computers operate in parallel in real-time and are controlled and time synchronized from a single computer program supervisor/executive. The executive directs the program execution and established priorities. The simulator accepts control inputs from the trainee via cockpit controls, other crew station controls or from the instructor operator station; performs a real-time solution of the simulator mathematical mode; and provides outputs necessary to accurately represent the static and dynamic benavior of the real world system within specified tolerance and performance

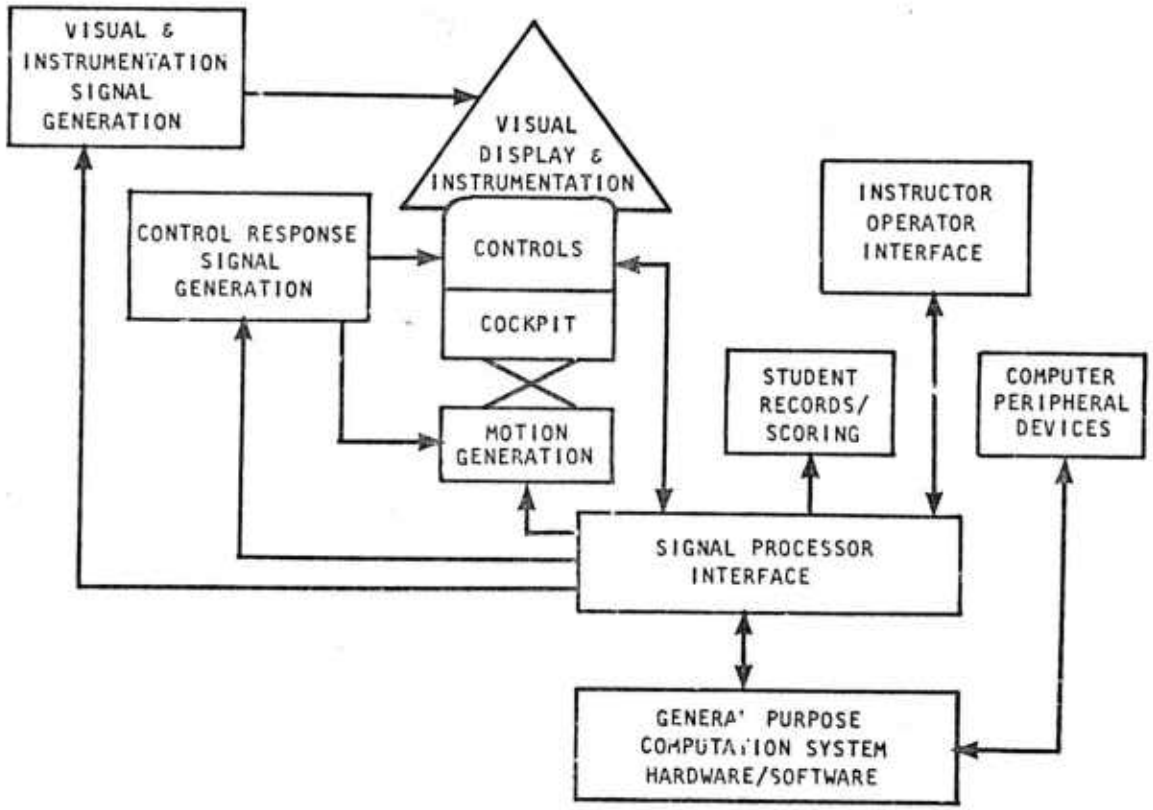

Figure 1.3-1. Typical Crew Training Simulator

criteria. Since training simulators are a combination of interdependent hardware and software, a joint development effort is required. As the complexity of training simulators 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 effective and efficient simulator procurement.

1.3.2 Automatic Test Equipment Characteristics

ATE consists of electronic devices capable of automatically or semi-automatically generating and independently furnishing programmed stimuli, measuring selected parameters of an item being tested and making a comparison to accept or reject the measured values in accordance with predetermined limits. ATE is used in place of manual devices, either because it is more cost effective, or the item being tested requires the speed and timing 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 will be found in one form or another in the majority of ATE systems. The controls and displays section consists of the computer and peripheral devices like control panels, magnetic tape cassettes or disks, a CRT and keyboard, and usually a small printer. The computer, as controlled by software, performs the tasks of operating the peripheral devices, switching test stimuli on and off, and measuring and comparing responses of the unit under test (UUT) to predetermined values. The operator will maintain ultimate control of the testing process through some of 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. The interface equipment may be a combination of hardware and software. It is normally designed to allow a single configuration of ATE to be used for testing several articles of system equipment. The maintenance level being supported by the ATE is determined by logistic planning.

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. In some cases (not always), arbitrary function generation and complicated wave analysis is also accomplished by software.

1.4 GUIDEBOOK ORGANIZATION

Because SWQA is a multi-phased, multi-organizational effort, the guidebook is organized to follow a logical software development flow within a typical ATE or TS system acquisition process. It is noted that not all ATE and TS system acquisitions are typical. Some may have fewer formal phases of development than others. Some are separate contracts. Others are contracted for as changes to prime weapon system contracts. Some acquisitions may have complex contracting arrangements involving multiple associate contractors. All such acquisitions however, follow some variation of the basic phases (Analysis, Design, Code and Checkout, Test and Integration, Installation, Operation and Support) in their development.

In discussing SWQA methodology in sequential fashion, considerations are presented unique to ATE and TS software development. The inter-relationship of PO and contractor responsibilities is discussed in Section 3.0; followed by treatment of software QA tasks as they are encountered in a logical software development sequence, in Sections 4.0 through 6.0. Sections 7.0 through 11.0

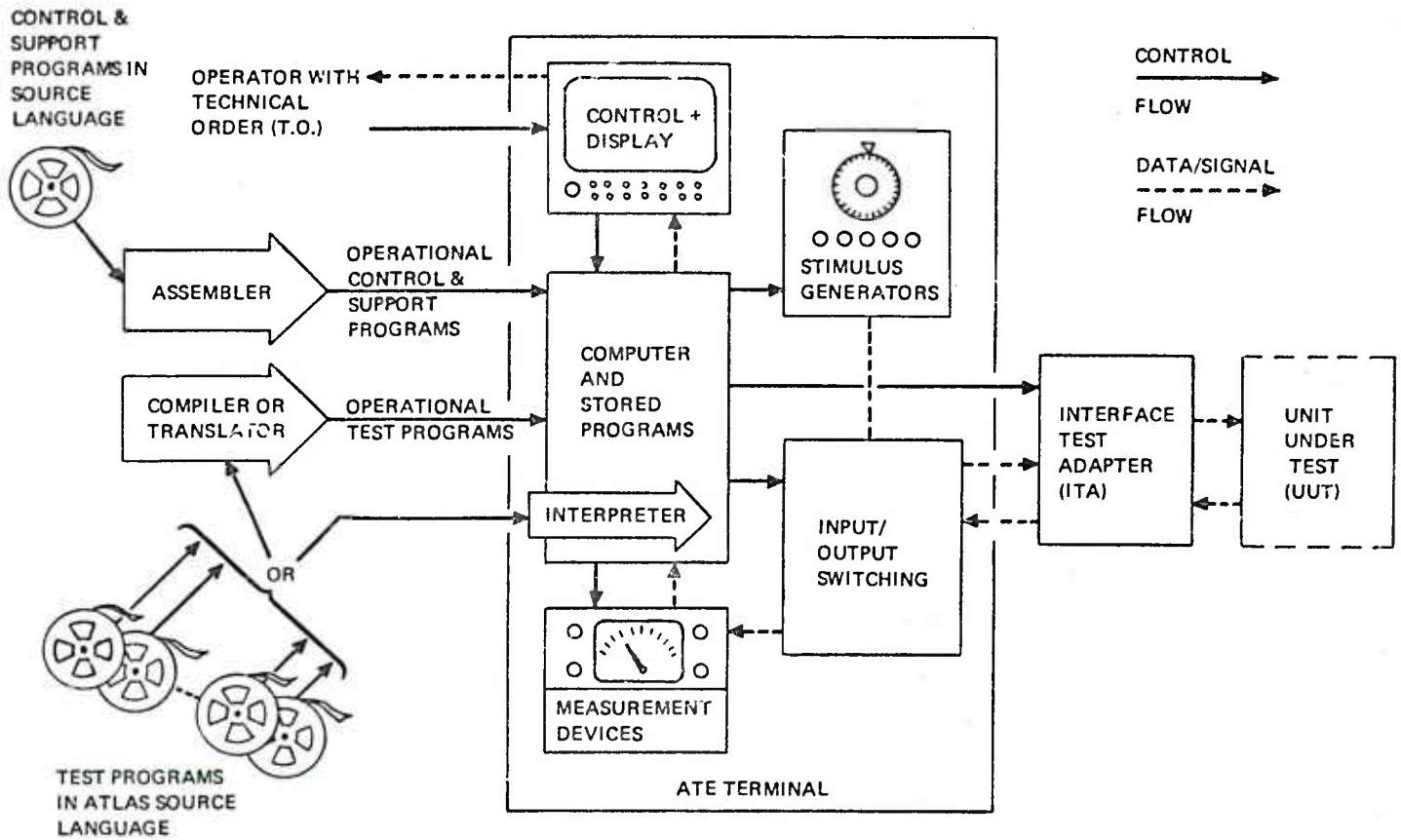

Figure 1.3-2. Typical ATE Configuration

include a bibliography, topic vs. Government document matrix, glossary, abbreviations and acronyms list, and a subject index respectively.

Section 2.0 APPLICABLE DOCUMENTS

The following is a list of government and industry publications relevant to software quality assurance, ATE and TS.

AFR 800-14, Acquisition and Support Procedures for Computer Resources in Systems

MIL-Q-9858A, Quality Program Requirements

MIL-S-52779, Software Quality Assurance Program Requirements

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

D180-19846-1, Software Quality Assurance Guide

MIL-D-83468, Digital Computational System for Real-Time Training Simulators

AFLC Regulation 66-37, Management of Automated Test Systems

SAMSO-STD-73-5B, Quality Assurance Program Requirements for Space and Missile Systems

MIL-STD-483, Military Standard Configfiguration Management Practices for Systems, Equipment, Munition and Computer Programs

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

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

DOD 4120.17-M. Strategic/Automated Data Systems

DOD Weapon Systems SW Management. Study, May 1975 Johns Hopkins University, Applied Physics Lab

Defense Systems Software Management Plan, March 19, 1976, Barry De Roze, OASD

Section 3.0 EARLY PLANNING CONSIDERATIONS

To plan adequately for ATE and TS software QA, the USAF PO must consider decisions and directions established early in the the conceptual phase of the prime weapon system acquisition cycle. Since ATE and TS are support systems usually contracted for separately, or as an addition to the prime contract, at same point during the full scale development phase of the prime system, much of the software development methodology, including SWQA will have been established at the time of ATE/TS contract award. Usually, QA and Configuration Management (CM) methodology similar to that applied to the prime weapon system, will also be applied to the ground support systems. The guidebook on "Computer Program Documentation Rquirements" CDRL item A009, discusses ways in which ATE and TS systems are acquired. Depending upon the particular contracting arrangement, early development planning may be done by the PO or by the contractor. Whichever applies, QA planners (within the PO or within the contractor's organization) should support early system planning efforts recognizing that methods of CPCI development established for the prime weapon ystem may influence those applied to ATE and TS systems.

3.1 ASSIGNMENT OF RESPONSIBILITIES

While the majority of the Quality Assurance efforts required for the ATE/TS software development life cycle are accomplished by the contractor, QA personnel within the PO must be among the first concerned with QA planning. Before software development even begins, much can be done toward defining quality software in the analysis phase. The following sections discuss the coordinated efforts of government and contractor QA personnel as a function of acquisition phase. Figure 3.1-1 illustrates the major USAF and contractor SWQA responsibilities as a function of ATE/TS system acquisition phase. Note that for ATE systems the contractor has the major task in preparing system and CPCI development specifications.

3.2 ORGANIZING FOR SWQA

Since the contractor performs the majority of ATE and TS production SWQA tasks, one of the PO's prime concerns in planning should be to evaluate how the contractor is planning and organizing for SWQA. Contractors should employ some kind of organizational checks and balances to promote an objective evaluation of software design. For example, a separate group should exist to evaluate and test computer programs. This group is charged with the responsibility for design and performance verification. Another separate group should exist to manage program configurations. This group is responsible for software identification, change control, problem resolution methods, computer program library controls, and release of all engineering documentation configuring software. Figure 3.2-1 represents one way of organizing for SWQA. In this arrangement, prime software development responsibility lies within the engineering organization during development phases (Analysis and Design) while quality assurance assumes prime responsibility for configuration control and acceptance during validation phases (test and integration and subsequent phases). This responsbility transfer is necessary because QA is inherently responsible for producing official objective records of requirements satisfaction. They must show exactly what programs were tested to what procedures in satisfaction of what requirements. Prior to this time, software engineering is responsible for conceiving, developing, and configuring a design up to a point where it is deemed ready for formal, controlled testing.

	Analysis	Design	Code and Checkout	Test and Integration	Installation	Operation and Support
AIR FORCE QA	PDR Review ● RFP ● CPDP ● TS Sys. Spec. ● SWQA Plan	CDR Review ● Test Plans ● Prelim. Part II CPCI Specs.	Audit Contractor Plans for ● CM ● SWQA Participate in CDR/FCA/PCA Review CPCI Part II Specs.			FCA PCA
CONTRACTOR QA	Prepare ● SWQA Plan Assist Prep. ● RFP ● SOW ● CPDP ● Dev. Spec. Support ● Source Selection ● RFP Evaluation	Review and assist Preparation of ● Prelim. Part II CPCI Specs. ● Test Plans ● Test Procs. ● SW Standards ● TRD's ● ICD's	Audit Controls of ● CPDP ● SWQA Plan ● CMP ● CP Library Review for QA Adequacy ● CPCI Part II Specs. ● VDD's ● V/V Plans ● Test Procedures ● Changes ECP's CCP's	Control Conduct of SW Tests ● Verify configuration ● Verify Set-up ● Planning Orders ● Discrep. Reporting ● Chge Accting ● Media Control ● Test Records ● Witness		

Figure 3.1-1. ATE/TS System Life Cycle SWQA Responsibilities

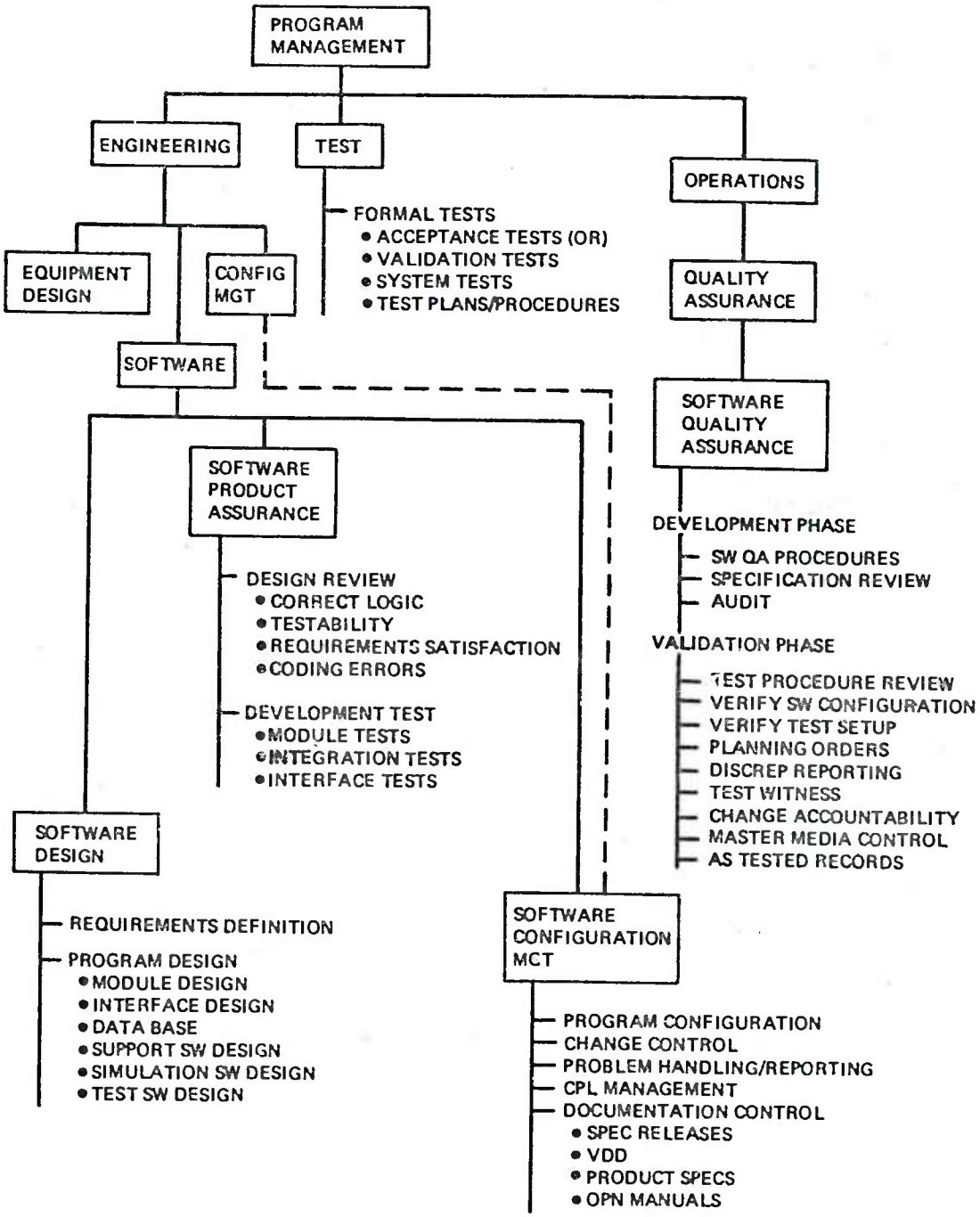

Figure 3.2-1. Typical SWQA Organizational Responsibilities

The program QA organization shown in Figure 3.2-1 includes a subset software QA function, which draws on the resources of other organizations in accomplishing program SWQA requirements. The software QA function should be staffed by one or more QA engineers, preferably knowledgeable in both quality and computer sciences, who will integrate and oversee the software QA effort for the project. The software QA engineer writes the SWQA plan for the project, reviews and audits documentation releases, audits requirements to procedures traceability, and insures the smooth integration of all other SWQA functions which require the resources of other organizations. He supports all program milestones - Design Reviews, Audits Test Readiness Reviews, Program Review meetings, etc., and is essentially a technical QA staff assistant, acting as consultant on software matters to QA management and factory QA personnel.

Variations of the organization shown in Figure 3.2-1 will exist for ATE and TS as necessary to integrate the responsibilities of UUT contractors, the ATE contractor, and weapon system design organizations interfacing with TS system designers. The key objectives in organizing for SWQA are twofold:

a. To accomplish an objective evaluation of performance independent of that performed by the design organization.

b. To arrange for control of program materials and insure that documentation conforms to contractual requirements and established programming standards and conventions.

The following sections describe SWQA planning activities as they would begin with contractor receipt of go-ahead for the analysis phase of a ground system.

Section 4.0 ANALYSIS PHASE

In conceiving QA policy for a ground system acquisition, QA personnel within the PO are guided by AFR 74-1 and AFSCR 74-6. These regulations refer to specifications such as MIL-Q-9858 and MIL-C-45662 for Quality Assurance. These specifications are not software oriented. Rather they define QA system requirements for any deliverable product and are imposed on the prime weapon system contract. MIL-S-52799, Software Quality Assurance Program Requirements, is gaining increasing utilization as a contractual requirement on large embedded software acquisitions, and is most appropriate for ATE and TS software acquisitions. Prospective contractors, however, frequently claim that establishing a software QA system, specifically compliant with MIL-S-52799, increases acquisition costs. This claim may have validity in certain cases. For example, in ATE acquisitions, operating system software is procured as part of the ATE system in a high percentage of its final deliverable configuration from the ATE contractor. A requirement imposing MIL-S-52799 on a contractor supplying proven off-shelf software, could be unnecessarily costly. The specification is totally appropriate, however, for the development of UUT software and the total integrated software package. It is for such reasons that QA planners should become familiar with system requirements and their impact on software as early as possible.

4.1 SYSTEM LEVEL REVIEW

The analysis phase consists of conducting the analyses and trade studies leading to preparation and approval of the computer program development specifications. Preliminary design of CPCI's is done concurrently in the latter part of the analysis phase and is concluded at the Preliminary Design Review (PDR).

4.1.1 System Analysis

System analysis consists of the hierarchial decomposition of system operational requirements and the allocation of these requirements to software, hardware and operator functions. These activities include the preparation of functional flow (or decision logic) diagrams, top-down design diagrams, system architecture and hardware/software risk, benefit and cost trades.

Functional flow diagrams consist of detailed logic network and work descriptions of each major function to be performed, in the order in which it is performed. The analysis of these diagrams, in conjunction with the frequency at which the functions must be performed, will be used in the determination of requirements for concurrent operations, critical timing relationship between functions and the allocation of processing resources to perform the functions. Decision logic diagrams contain the same information as functional flow diagrams with the addition of decision information.

Hardware/software trade studies are conducted to ensure the proper allocation of functions to equipment and computer programs. These risks and cost trades address design to cost, growth potential, development, integration and schedule factors. System modeling and simulation are used to support these trades and verify the results. Pertinent trade studies are documented and included in Subsystem Design Analysis Reports submitted at the end of the phase.

4.1.2 Computer Program Requirements Analysis

Requirements analysis is an iterative process, repeated with revised requirements until a compatible set is pro-

duced. A disciplined and systematic documentation system and control of changes is required in order to integrate and correlate the work among participants and throughout the iterations.

Functional/performance requirements are derived from the requirements allocated to software from system analysis. These derived requirements are the result of software from hierarchial decomposition from the very general requirements to those sufficiently detailed to provide the basis for design and verification. The highest levels will be restatements of the allocated system specification requirements. These requirements are further decomposed and allocated to CPCI. Lower levels are derived to take the form of text, graphs, charts, etc., and include performance, timing and environment considerations. The result is the computer program development specification which consists of a set of statements that specify exactly what the CPCI will do, how well it will do it, and the conditions under which it must perform. Demonstrating that the computer program satisfies each of these requirements is the basis for its acceptance.

The documentation technique includes specific means to easily trace any element through its parent element to the top of the model. Graphic tools that will be used in decomposition include simple tree structures showing the functional breakdown into more detailed requirements and Hierarchial-Input-Process-Output (HIPO) charts that show the inputs, processing and outputs for each element of the decomposition.

The analysis documentation describes critical decision points encountered, and provide the rationale supporting the adopted approach, e.g., trade studies, simulations, modeling, etc. It is expected that conflicts among requirements will be exposed by the analysis process. A list of all conflicts and alternatives are prepared. The analysis shows how the conflicts arose, what requirements must be relaxed or modified, and by how much, in order to relieve the conflict.

4.1.3 SWQA Responsibilities

SWQA personnel should evaluate the functional and verification requirements of the computer program development specification during the development/analysis effort. SWQA should review the requirements in view of the total system requirements, evaluate the ability of the requirements to be verified and evaluate the adequacy of the verification requirements. SWQA should assure that complete, consistent, unambiguous, testable and up-to-date requirements are established in order to insure the success and quality of the software development effort.

SWQA personnel should prepare a checklist to cover the surveillance of the software requirements analysis phase. This includes a list of criteria for specific items to be evaluated, such as the following:

a. Is a development specification for each CPCI prepared?

b. Docs the specification tell what the computer program must do, how well, and under what conditions and describe the environment in which it is to operate?

c. Are requirements stated singularly, clearly and concisely and could a reader understand the specification even though he is not completely familiar with terminology peculiar to computing engineering?

d. Are all requirements traceable to, or derived from, a higher level specification, and is the source of each requirement or derivation shown?

e. Are performance requirements for each function described in separate para-

graphs? Do these paragraphs include source and type of inputs, and destination and type of outputs?

f. Is each requirement identified, stated separately, and in a manner that can be verified by test, analysis or observation?

g. Are quantitative terms, e.g., ranges, accuracies, tolerances, rates, bondary values, and limits used in stating requirements, such as a specific output that can be recorded as evidence of satisfaction?

h. Is the specification limited to defining requirements, and free of design solutions or methods to satisfy these requirements, stating what should be done rather than how it will be done?

i. Are all operational interfaces with the computer program, including both hardware and software, identified including references to those specifications which define those interfaces?

j. Are all applicable non-operational interfaces related to computer program support and code generation identified; such as specific programming language, compiler, data base management system, loaders, other utility programs, or unique support hardware? Are references to appropriate documentation of these interfaces identified?

k. Are all inputs to and outputs from the computer program identified?

l. Are the specified requirements free of material that is not intended to be contractually binding?

m. Do the quality assurance provisions specify the method to be used to verify each performance requirement in sufficient depth to provide the basis and scope of the test plan?

n. Are the methods to be used and evidence to be obtained to verify compliance appropriate to the requirement being verified?

In accomplishing ATE/TS system specification reviews, QA personnel should look for ambiguities, omissions and conflicts which make the job of defining the CPCI difficult. Clarity and quantification of the following factors should be examined.

a. Definition of interfaces and boundaries between system segments.

b. Capabilities which are to be resident in hardware vs. software - (i.e., certain routines - arithmetic self-test, utilities, loaders, etc., might be resident in mini or micro-processors, in Read Only Memory (ROM) or in the main program).

c. Definition of measurable and verifiable performance goals; for example, in ATE system design - what are the design goals in terms of cost, throughput, maintenance etc.

d. System design commonality requirements among installations.

e. Software transportability (interchangeability among machines).

f. System capacities - data storage handling rates, stimuli/measurement capabilities, etc.

g. Deletion of unnecessary design constraints (i.e., specifying second vs. third generation ATE).

Although the above concerns are primarily System Requirements Review (SRR) and System Design Review (SDR) issues of concern to engineering organizations, it is necessary for the QA planner to review system requirements thoroughly so that the stringency of required software QA requirements can be properly assessed, and to provide clarifications and improvements helpful in CPCI development specification preparation.

4.2 ESTIMATING SWQA REQUIREMENTS

Early in the analysis phase, system designers will have allocated system requirements to preliminary CPCI development specifications. By now, QA planners should have a fair idea of the number, size, complexity, schedule and cost targets for CPCI's. Although traditionally not a QA consideration in designing system controls, cost is now a prime design driver in major weapon systems. QA planners must therefore justify the control methodology they select for a given set of software. Very complex systems, destined for multiple users, involving several hundred thousand instructions, multiple CPCI's, of relatively complex unproven software are prime candidates for the full-up disciplines of MIL-S-52779. Simpler systems require proportionately fewer controls.

Although recent committee actions and software symposiums are working on standardizing Software Configuration Management (SCM) and QA methods, there is no standard at this time for estimating the SWQA task. Figure 4.2-1 presents generalized guidelines for selection of QA control elements as a function of system size, complexity, maturity and schedule. After ascertaining the level of control stringency for QA requirements, the next step is the preparation of a preliminary SWQA plan.

4.3 PREPARING A SWQA PLAN

There is currently no established standard Data Item Description (DID) for a SWQA plan. However, a committee is meeting quarterly to establish a SCM standard including a DID for a SWQA plan. (See Bibliography reference 2). The Navy NELC, however, has developed a Software Quality Control Plan Data Item, UDI-R-11. A SWQA program is complementary to a SCM program. Together the two programs, documented by written plans, implement the Computer Program Development Plan (CPDP).

A typical SWQA plan outline based on MIL-S-52779 is presented in Fig. 4.3-1.

As stated previously, for some acquisitions, particularly ATE, the Operating System (OS) software package (control, support and test) may be purchased essentially "off-shelf" from the ATE contractor. It is normally inappropriate to impose a SWQA program embodying significant development controls on fully developed and proven software. The weapon system contractor and his associates, however, must integrate their software (UUT test and adapter programs) with the vendor supplied operating system to define a complete deliverable software package. The SWQA plan stringency, therefore, is CPCI dependent. An early determination should be made as to which programs are subject to which specific controls. The selected controls, including the formal SWQA plan should be included in the contracting vehicle (Contract Change Proposal (CCP), Engineering Change Proposal (ECP), Statement of Work (SOW), Contract Data Requirements List (CDRL)). The prime contract then passes along appropriate requirements to the ATE or TS system contractor. A brief discussion of these SWQA plan elements follows. In subsequent sections of this guidebook, adaptions and considerations unique to ATE and TS software are discussed for each of these general elements.

4.3.1 Work Tasking and Authorization Procedures

Procedures for defining and estimating resource requirements for SWQA tasks for described in this section. Job descriptions, responsibility assignments, schedules, completion dates, methods of authorizing work, progress and status reporting are identified in this section. A checklist of typical software QA functions by task is presented in Fig. 4.3-2.

4.3.2 Configuration Management

The SWQA plan complements the SCM plan by defining the quality assurance measures to be taken in accomplishing

	SIMPLE	MODERATE	COMPLEX
SYSTEM	• SINGLE CPCI/SEGMENT • SINGLE PROCESSOR • WELL DEFINED REQTS • HIGH % ASSY LANGUAGE • LOW COST • LESS THAN 25K INSTR • LESS THAN 5 PROGRAMS • BATCH OPERATION • MINIMAL I/O AND DISPLAY • DEVELOPED AND PROVEN O/S • SHORT PROJECT DURATION	• LESS THAN 5 CPCI'S • 75% HOL/25% MACHINE • MULTIPLE STATION CONFIGURATIONS • LIMITED I/O, DISPLAYS • LESS THAN 10 PROGRAMS • PARTIALLY DEFINED REQTS • LIMITED OPERATOR INTERVENTION • LESS THAN 100K INSTRUCTIONS	• MANY CPCI'S OR SEGMENTS • GREATER THAN 100K INSTRUCTIONS • MANY STATION CONFIGURATIONS • COMPLEX I/O AND DISPLAY • HIGH % HOL • MULTIPLE SOURCES OF REQUIREMENTS • MULTIPLE USERS • UNDEVELOPED OPERATING SYSTEM • COMPLEX SUPPORT SOFTWARE • EXTENSIVE PROJECT DURATION
	QA CONTROLS	QA CONTROLS	QA CONTROLS
QA CONTROLS	• DESIGN SURVEILLANCE • CONFIGURATION CONTROL • USER DOCUMENTATION • PROGRAM IDENTIFICATION • SW CERTIFICATION	• DESIGN SURVEILLANCE • WRITTEN SW QA PLAN • CONFIGURATION CONTROL • LIBRARY CONTROLS • PROBLEM REPORTING/CORRECTIVE ACTION • TEST/ACCEPTANCE RECORDS	• WRITTEN SW QA PLAN • WORK TASKING ESTIMATION • FORMAL SW QA ORGANIZATION • CONFIGURATION MANAGEMENT • DESIGN REVIEW AND ANALYSIS • LIBRARY CONTROLS • FORMAL TEST PROGRAM • DOCUMENTATION CONTROLS • REVIEWS AND AUDITS • TOOLS TECHNOLOGIES IDENTIFIED • MEDIA STORAGE HANDLING • FORMAL RECORDS

Figure 4.2-1. SWQA Requirements Versus System Complexity

SOFTWARE QA PLAN OUTLINE - REF. MIL-S-52779

I. Work Tasking and Authorization Procedures

- Procedures and Schedules
- Work Descriptions
- Status Reports
- Resources Estimates

II. Configuration Management

- Baseline Establishment
- Change Accountability
- Audits

III. Testing

- Analysis to Determine Testability
- Test Plan/Procedure Review
- Monitor and Certification of Test Results
- Tests vs. Requirements Traceability

IV. Corrective Action

- Problem Reporting and Measuring
- Trend Analysis
- Corrective Action Assessment

V. Library Controls

- Code Control and Related Documentation
- Media Identification and Protection
- Change Control

VI. Computer Program Design Review

- Contract Compliance
- Evaluation of Design Logic

VII. Software Documentation

VIII. Reviews and Audits

IX. Tools, Technologies, Methodologies

X. Subcontractor Controls

Figure 4.3-1. Typical SWQA Plan Outline

Software QA Functions
by Tasks

A. Software QA Planning for Total Project

1. Contract Review, Planning, Coordination
2. Preparation of Software QA Plan
3. Preparation of Project and QA Command Media
4. Review of Computer Program Development Plans (CPDC)
5. Review of Software Configuration Management Plans
6. Preparation of Configuration Audit Plan (FCS/PCS or FACI)
7. Preparation of Software QA Audit Plan and Procedures

B. Work Tasking and Authorization (Review and Monitor)

8. Work Authorization
9. Planning and Scheduling
10. Performance Measurement and Reporting

C. Software Standards

11. Design
12. Documentation
13. Programming

D. Software Documentation Reviews

14. Review Documentation Plans and Schedules including Configuration Control Methods
15. SIRD (Software Interface Requirements Document)/TRD's
16. Development (Requirement) Specifications
17. Product (Design) Specifications
18. Data Base Specifications
19. Interface Specifications
20. Interface Control Documentation
21. Configuration Drawing
22. Process (System Generation) Document
23. User Manual
24. Maintenance Manual
25. Version Description Document

E. Software Design Surveillance

26. Requirements Analysis
27. Preliminary Design
28. Detail Design

F. Software Design Reviews

29. Software Requirements Review (SWRR)
30. Preliminary Design Review (PDR)
31. Critical Design Review (CDR)

Figure 4.3-2. Typical SWQA Task Breakdown (Sheet 1 of 2)

G. Software Verification and Test

32. Review of Verification and Test Plan
33. Review Test Procedures
34. Track Informal Tests
35. Attend Pretest Meeting
36. Technical Support of Formal Tests
37. Review of Test Supports

H. Software Tools (Review and Control)

38. Identify and Evaluate Automatic Test Tools
39. Tool Validation
40. Configuration Control of Tools

I. Coding Control (vs. Standards)

41. Review Program Listings

J. Software Configuration Management

42. Audit Compliance to Software Configuration Management Procedures and Effectiveness of the Program

K. Computer Program Library and Media Control

43. QA Establishment and Maintenance of CPL
or 44. QA and Engineering Establishment and Maintenance of CPL
or 45. Engineering Establishment and Maintenance of CPL with QA Audit CPL
46. Establish Methods and Procedures for Media Control
47. Audit Implementation and Control of Media Control

L. Discrepancy Reporting, Change Board and Corrective action

48. Support Change Board Actions
49. Discrepancy Reporting - Tracking, Analysis and Corrective Action

M. Configuration Audits

50. Pre-Audit Reviews
51. Formal Qualification Review (FQR)
52. Functional Configuration Audit (FCA)
53. Physical Configuration Audit (PCA)

N. Software Quality Assurance Audits

54. Audit Software Controls (In-house)
55. Audit Software Controls (Remote Sites)

O. Procured Software

56. Participation in Source Selection
57. Participation in Pre-award Review and Survey
58. Review of Procurement Specifications
59. Review of Subcontractor Software QA Plan

Figure 4.3-2. Typical SWQA Task Breakdown (Sheet 2 of 2)

change accountability (i.e., assuring the change has been incorporated in all applicable media and properly retested and reconfigured). While overall responsibility for SCM should be vested in a separate group, it is a QA responsibility to assure the "as built" meets the "as designed" and to provide appropriate records. It is also a QA responsibility to audit the SCM function.

4.3.3 Testing

The software QA plan should provide for an independent and objective evaluation of the CPCI design. This may be accomplished by design analysis, (review of program design, logic, code), inspection by compliance with programming standards and by actual test. This section of the QA plan addresses design verification by test, and how responsibility for such testing is vested organizationally. Developmental vs. formal (acceptance or validation) testing is distinguished. With respect to testing, therefore, the software QA program should provide the following:

a. Design analysis to determine testability

b. Test requirements, test plan, and test procedure review

c. Witnessing and controlling of formal testing

d. Documentation allowing repeatability of tests

e. Identification and acceptability of support software

f. Record keeping

4.3.4 Corrective Action

The software QA program should provide for prompt detection and correction of software discrepancies. Corrective action includes categorization of software discrepancies, reporting, trend analysis, and corrective action assessment.

4.3.5 Computer Program Library Controls

The software QA plan should provide a controlled repository for all programming materials, media and documentation. This includes decks, tapes, listings, flow charts, all configuration descriptors, change records, problem reports and summaries. This library should be maintained and secured to prevent unauthorized changes to controlled documentation and media.

4.3.6 Design Review

The software QA plan should provide a function which assures that program design is well structured and efficient, is verifiable, satisfies the functional requirements completely and complies with contractual programming standards and conventions. The organization in which responsibility for these review functions is vested should be defined.

4.3.7 Software Documentation

The software QA plan must provide for methods of assuring complete and correct documentation describing the software CPCI. The quality assurance plan should assure that all documentation describing software requirements, configuration, tests and analysis, user documents, change and problem records are available prior to delivery with the CPCI.

4.3.8 Reviews and Audits

The software QA plan should define QA conduct of, and participation in, a series of planned periodic audits to assure compliance with approved contractor plans, procedures and standards. Both formal reviews (Preliminary Design Review (PDR), Critical Design Review (CDR), Physical Configuration Audit (PCA), Functional Configuration Audit (FCA)) and informal (i.e. periodic audits of compliance to approved CPDP, SWQA plan, CM plan, etc.) audits should be addressed.

4.3.9 Tools, Technologies and Methods

The software QA program should describe the use of any special tools or technologies used in the development and test of computer programs. Analyzers, automatic test program generators, special language translators, configuration control aids, etc. are among the software tools which should be addressed in this section.

4.4 PREPARING SWQA REQUIREMENTS FOR THE RFP

A well prepared SWQA plan should cover all the quality concerns applicable to contracting for ATE and TS software. However, there are numerous other RFP elements which impact QA in addition to the SWQA plan. One of the PO's prime concerns during the analysis phase should be to assure that only necessary and cost effective SWQA requirements are imposed in the contract. To do this he must first estimate complexity, criticality and usage requirements for the CPCI's involved and then consider the numerous RFP elements which affect SWQA. Among those most applicable to QA are:

a. Procurement Specification. (QA provisions, Section 4.0)

b. Statement of Work (SOW)

c. CDRL

(1) Computer Program Development Plan
(2) Configuration Management Plan
(3) Configuration Index
(4) Version Description Document (VDD)
(5) Drawing and Specification Index
(6) Acceptance Test Procedure
(7) Delivery Data Package
(8) Deviations and Waivers
(9) User Manual

d. Contract Terms and Conditions/ Special Provisions

(1) QA System Requirements
(2) Warranty/Rejection Clause
(3) Inspection at Source Clause

Many functional disciplines contribute to the synthesis of an RFP. Each assures its area of concern is adequately covered. A common pitfall in RFP preparation is the inclusion of redundant or conflicting requirements. For example, the subject of "testing" may be covered in a variety of places such as:

a. Data Item Description (DID) for CPDP

b. DID's for Test Plans/Procedures

c. Section 4 of CPCI Specifications

d. Acceptance Test Section of the SOW

Conflicts in test planning, procedduralizing, establishing prerequisites, and reporting data can and do exist within RFP elements such as the above. For example, the DID for a CPDP may require system level integration testing of CPCI's as a prerequisite to CPCI formal acceptance test whereas the Quality Assurance Provisions (Section 4 of CPCI specification) may not contain a similar requirement. Careful review of each RFP element for consistency with other RFP elements is required. One way of minimizing conflicting RFP requirements is as follows:

a. Identify the needed QA control elements for subject acquisition. (See Fig. 4.2-1)

b. For each control element (i.e., configuration management) identify by title and number all RFP elements which address that subject.

c. Create a cross reference matrix of QA control elements vs. RFP elements.

d. Review, edit, rewrite or delete conflicting or redundant RFP elements as required.

e. Coordinate recommended changes with applicable functional organizations.

Frequently RFP preparation is a hastily accomplished activity, especially in ATE or TS acquisition. This is mainly due to the fact that the people most qualified to work on the RFP have on-going responsibilities for the primary weapon system. As a consequence, RFP work must be interlaced with these other duties. Coordination and dedication to the RFP task, therefore, suffers and results in poorly prepared requirements. Exercises such as described above may take more of a "front end" investment, however, it is well worth the effort to prevent costly consequences of ambiguous, redundant or conflicting requirements.

4.5 SUBCONTRACTOR CONTROL

Paragraph 4.2, "Estimating SWQA Requirements" discuss in depth the methods of determining the stringency of SWQA requirements to be imposed when contracting for software acquisition. Fig 4.2-1 presents a guide for selecting disciplines to impose upon a given subcontractor commensurate with the size, cost, schedule, complexity and manageability of various software components. Good subcontracting practices take those issues into consideration, resulting in imposition of only practical and necessary controls. In a prime contract requiring MIL-S-52779, the contractor should flow down to his software subcontractors only those work elements needed to achieve overall control of the end item. As mentioned previously, it would be unwise (costly) to demand a compiete re-structuring of a vendor's software organization and methods solely for the purpose of complying with MIL-S-52779 if he is supplying an essentially off-shelf proven product.

Section 5.0 DESIGN PHASE

The design phase begins with CPCI development specification (Part I) approval at PDR and concludes with Part II specification approval at CDR. The CPCI development spec is used as the allocated (functional and performance) baseline against which preliminary design and validation test planning can commence. The reader is referred to the "Computer Program Documentation Requirements" guidebook for a discussion of development specifications for ATE and TS software. It is noted that a development specification is not necessarily applicable to test, control and support software for ATE. The guidelines presented below, however, assume that a CPCI development specification is required for both ATE and TS software as the basis for design, management controls, verification and configuration management.

Surveillance of the software requirements and design activities, will be performed by QA personnel during the software development effort, to provide an independent evaluation of the tasks performed with technical review feedback to the designers. In addition, this surveillance will provide a level of understanding of the software requirements and design necessary to evaluate detailed test plans and procedures. Accordingly this section deals with QA activities during the design phase, primarily design review tasks supplemented by a discussion of software standards and design and development tasks.

5.1 DESIGN REVIEW AND SURVEILLANCE

Design Review activities by QA personnel are conducted for a variety of reasons.

a. During the analysis phase, QA reviews are conducted to assure that system level specifications provide a complete, accurate and unambiguous definition of system and software performance requirements. During this phase the QA review assures that the allocation of system requirements of CPCI's is clear and complete; and does not require assumptions or further research to determine what the CPCI must do.

b. During the preliminary design phase, quality reviews are conducted to insure that the decomposition of the CPCI into subfunctions is done in an orderly, clearly defined, manageable and verifiable manner.

c. During the detail design phase, the quality review is concerned with examining program design for compliance with functional requirements and programming standards.

d. During the Test and Integration phases, QA activities expand to include not only reviews of test plans and procedures, but also include physical methods for managing test conduct and protecting software configuration integrity.

The following paragraphs discuss design review guidelines applicable to the period between PDR and CDR.

5.1.1 Preliminary Design Review Activities

Preliminary design utilizes the concept of hierarchial decomposition. The higher level requirements for each CPCI are broken down into their major functional modules. The basic structure of the computer program is defined including the methods of sequencing and controlling module execution. The communications protocol between modules is designed, modeled and analyzed. A preliminary definition of the data base structure, organization and content, is generated. Preliminary sizing and timing estimates are prepared for each module. The management of these computer resources is a continuing effort that should be addressed throughout the design and implementation of each CPCI.

Modeling of time-critical functions is also continued throughout the design phase. Detailed design of external interfaces is begun and preliminary interface control drawings (ICDs) are prepared to describe data rates, interrupt frequencies, data formats, and limits.

5.1.1.1 Documents Generated. Documents resulting from the preliminary design process are as follows:

a. A preliminary product (Part II) specification.

b. A design control document which contains the overall software design describing the interrelation and control between CPCIs.

c. A test plan covering testing at all levels, including methods, locations, equipment, personnel and preliminary sequencing/scheduling.

d. Preliminary ICDs.

e. Timing and sizing data.

f. A Test Requirements Document (TRD) for each CPCI (ATE test software).

g. Computer program development plan update.

h. User's Manual (Draft).

i. Design and Coding Standards.

5.1.1.2 SWQA Responsibilities. During preliminary design, the basic design for each CPCI is established. Software QA personnel will prepare a checklist to cover surveillance of the preliminary design phase. This will include criteria for specific items to be evaluated, such as the following:

a. Are the functions allocated to each CPCI further allocated to the modules comprising the CPCI?

b. Are the methods of sequencing and controlling module execution and the communication protocol between modules designed and analyzed?

c. Are time and core space allocated to individual modules?

d. Is the UUT stimulus, test sequence, ancillary data been specified?

e. Have the UUT diagnostic capabilities been specified?

f. Has the total UUT capability been decomposed into subfunctions?

g. Does the TRD meet CPCI development specification requirements?

h. Are ATE/TS self test calibration frequencies/tolerances specified?

i. Are functional data and control flow diagrams, within and among with CPCIs, completed?

j. Is a preliminary definition of the data base structure, organization, and content generated?

k. Are time-critical functions analyzed to verify that system operations timing requirements can be met?

l. Is the development and documentation of all required computational algorithms completed?

m. Is a functional description and preliminary design for each module prepared?

n. Are timing and sizing summaries prepared and periodically updated, and do they provide allowances for growth?

o. Are the standard naming, interface, etc., conventions followed?

p. Is the basic design compliant with the project design standards?

q. Is the design compatible with external interfaces?

r. Are data handling limitations of the hardware and software clearly published?

s. Are the requirements for ATE test software properly decomposed and quantified? (i.e., requirements for self test, operational assurance/fault isolation, calibration?)

t. Are the support software requirements properly referenced for execution of CPCI designs?

u. Are external interfaces well defined (control, support, test)?

5.1.2 Detailed Design Review Activities

The detailed design phase will normally begin after a successful preliminary design review. Although the emphasis changes from requirements analysis to software design, all personnel and analysis/documentation systems remain intact. If at some point in the design, coding, or test and integration, it becomes necessary to redefine/ redistribute the requirements, the change can be more easily implemented by processing it through the previous sequence of development steps to arrive at the current version with complete and updated documentation.

Simple design of program components and relationships between modules are highly desirable. Ideally, program components should be modularized on the basis of performing a single logical transformation (e.g., compute SIN X) or on the basis of the control structure organization. The simplest method of passing data between program components is to pass only the data needed directly to the component in a calling sequence. In a large complex program, this can result in inefficiency and redundancy. The use of common data structures and common control blocks where the required data is a part of a larger collection may provide a more efficient relationship between modules. In this case, the design of the common data base is an integral part of the design process. During the detailed design phase, the software and data base design tasks are completed and documented in sufficient depth to permit coding. Concurrent with these tasks, the initial test plan document is updated, and detailed test procedures are developed and documented.

5.1.2.1 Requirements Decomposition. The detailed design process consists of the decomposition of computer program requirements into progressively lower level functions until the design is completed and documented in sufficient depth to permit coding to begin. Coding takes place only after a successful CDR. This is not to say that the entire design must be completed before coding begins, since the CDR may be conducted incrementally for completed design segments. Certain functions of the CPCI design may require early implementation; therefore, design of these functions can be completed early and scheduled incrementally for CDR.

5.1.2.2 Program Modeling. The design process also includes modeling of design features that pose difficult design problems and critical marginal timing. This technique utilizes a simulation of the computer program representative of the computer loading, timing and execution sequences and pilot programming of the troublesome design solution. A proposed solution may affect the computer program functional/performance requirements that were generated during the analysis, requiring a change in the requirements.

5.1.2.3 Software System Models. During the design phase, system models continue to be developed in a hierarchial manner. Initially, emphasis is placed on development of two separate models detailing (1) computer processes, and (2) data elements. The models are prepared in the strict, structured format previously

defined. A most important feature of these models is, that although separate, they are complimentary and can be easily cross-referenced. This facilitates a self-check of the design process which may be the technique that exposes design conflicts.

5.1.2.4 Data Base Design. Design of the common data base must be completed and reviewed early in the design phase in order that it may be coded and verified on a schedule that will enable the use of that data base to support testing of the first modules to complete the coding phase. The data base design schedule must be consistent with, and support the scheduled implementation of, program modules. The design must show types of storage, access methods, structure and precise definition of each element of the data base.

5.1.2.5 SWQA Responsibilities. SWQA personnel evaluate the detailed design of the computer program and data base. SWQA prepares a checklist to cover surveillance of the detail design phase, including criteria for specific items to be evaluated, such as the following:

a. Are module functions allocated to the lowest level of individually identifiable computer program components?

b. Is a detailed design representation, from which the software will be coded and debugged, prepared? Sufficient detail to permit direct translation to computer program code and to derive requirements for development testing of design is required.

c. Is the refinement of data storage and timing allocations completed?

d. Is the detailed definition of the common data base content, structure, and control completed?

e. Are methods of accessing data base elements defined?

f. Is the data base design consistent with the basic design?

g. Is the identification, design, and specification of parameters, passed, entries, and normal and abnormal exits for all common subroutines completed?

h. Is the complete detailed definition and documentation of software internal and external interfaces completed? (i.e., between ATE control, test, support software)

i. Is the test plan updated to reflect any changes in test schedules, test facilities, and test requirements resulting from information developed during detailed design?

j. Are all TRD's complete?

k. Is the detail design compliant with project design standards?

l. Does the design contain provisions for debug, test and maintenance?

m. Are all the sources of ATE test software requirements being considered in the preliminary Part II specification design (i.e., -end-to-end, diagnostic, station self test, TRD data, circuit analyses, etc.).

n. Do preliminary Part II CPCI specifications contain a matrix showing traceability of software requirements to modes of verification.?

5.1.2.6 Final Considerations. The completion of draft product specifications, ICD's, test plans and TRD's, signals the actual end of the design phase. Theoretically all of the foregoing should be available at CDR. However, much is frequently incomplete at the scheduled CDR date. The use of positive software development tracking methods (refer to guidebook on Measuring and Reporting Software Status) will minimize the chances of reaching CDR with less than expected progress in software definition. Ideally, the draft CPCI product baseline, less

listings, is approved at CDR. Theoretically, if the basic program design (narrative and flow) is sound, a mechanical task of code and debug is all that remains. Unfortunately both ATE and TS software are subjected to a perpetually changing requirements environment. As weapon system features and UUT capabilities change, so does the ground system software. A pitfall resulting from this environment is that original requirements documents become old and obsolete; designers are forced to design to change paper - engineering change orders, revision notices, etc. Updating of requirements documents sometimes suffers. The consequence is, that instead of a given CPCI version satisfying a given CPCI Part II Section 3.0, several versions exist, all of which satisfy the Part I, but each of which satisfies a different set of changes to the Part II specification. The specific program design then, may not necessarily be defined in Part II but rather in myriad of changes to it; to the ICD, to the TRD and other ancillary "design to" documents, which must be updated to FCA. Unless QA personnel are aware of this pitfall, that aspect of software configuration control dealing with requirement satisfaction may be at worst, completely violated, or at best be a monumental task to sort out. It is therefore important at this phase of the project (completion of design phase at CDR) to assess the traceability of fundamental system specification requirements through the prime item development specifications, CPCI development and draft CPCI product specifications.

5.2 SOFTWARE STANDARDS

Another item of major importance to SWQA during the design phase is concern for software standardization. Uniform identification, terminology, specification format, methods of coding, flow charting, annotating listings, reporting data, etc., all fall into the category of software standards. Other guidebooks deal extensively with documentation standards which address specification structure and format as required by MIL-STD-483, 490 and others. In this section the subject of standards will be continued to include design, coding and flow charting techniques and conventions as necessary to achieve understability and uniformity in software design descriptions. The standards govern the work of all design personnel in the software development organization. Although UUT test programs will be written in relatively straightforward test language, the basic programming standards apply to the development of total test package software as well as to TS software. The USAF acquisition engineer should ensure that these standards are reflected in his procurement specifications and are implemented by the winning contractors.

5.2.1 Design Standards

Some of the design standards which should be provided are outlined in the following paragraphs:

5.2.1.1 Hierarchial Program Design. Programs should be designed in a hierarchial manner, where the levels of the hierarchy correspond to levels of control of the tasks performed by the program. Each program proceeds from a single starting point and is broken down into a tree structure of computer program components. The top-level component contains the highest level of control logic and decisions within the program, and either passes control to the next level component or identifies the next level components for in-line inclusions. This decomposition of control and tasks continues to successively lower levels until all functions within the system are defined. The components of the program will be closed subprograms with a single entry and a single exit point, and of a size which can be reasonably comprehended and viewed.

5.2.1.2 Standardized Logic Structure. Logic structure should be standardized in a manner that enhances readability of programs and reduces intricate logic that is difficult to validate and

verify. Only closed logic structures should be employed in the construction of program components. Closed logic structures are logic structures that have a single entry point and a single exit point. (Logic structures employed in the software design, such as sequence logic diagrams, if-then-else logic diagrams, do while logic diagrams, etc., are specified in the standard.)

5.2.1.3 Readability. The readability and communicability of program routines are enhanced by restricting their size to that which can be easily viewed and comprehended. The maximum limit will be N (for example, N: 50) executable source statements. Exceptions are reviewed on a case-by-case basis.

5.2.1.4 Naming Conventions. Standard naming conventions are essential to understanding of the code by those responsible for its test and maintenance and for facilitating communications among the persons responsible for program development. Standard conventions should be identified and applied to the naming of variables, data base items, modules, and lower level computer program components. Names should be comprised of alphanumeric characters and are mnemonic so that it is possible to distinguish a computer program component's place in the hierarchical program structure by its name; common data base names are distinguishable from local variable names; and common subroutines are distinguishable from module unique subroutines.

5.2.1.5 Data Base Structure. The computer program data base consists of local parameters which are used only by a single program component and global parameters that are used by more than than one program component. Local parameters should be controlled exclusively by the program component. Global parameters should be maintained in a data base structure that can be supported by the programming language selected for the job. Data structures and access methods will be specified and all addressing of data base items will be performed only through the use of data labels.

5.2.1.6 Subprograms. Subprograms that are called from only one point or that operate upon the same data at each call should use the common data base to receive and pass parameters. Subroutines that operate on different data with different calls should receive and pass parameters through a calling sequence. Any error condition should be returned to the calling module via a parameter list and calling component handles the error condition.

5.2.1.7 Flow Charting. Flow charting standards are applied during the preliminary and detailed design phases of the software development process for the purpose of maintaining consistency among the various pieces of the software system and providing a means of understanding the proposed design. Normally included in the flow charting standard are: 1) flow charting procedure including direction of flow (top to bottom, left-to-right), level of detail, use of page connectors, etc.; 2) a standard symbology for each function, decision, connector, etc.; and 3) a standard nomenclature to be used in naming subrouting, constants, variables, etc. Analogous standards should be estalished for alternate design representations such as Program Design Language (PDL) if used.

5.2.1.8 Coding Language. A HOL should be selected and specified as the standard for coding of computer programs. The use of assembly or machine langua;e should be restricted to coding of time-critical routines, certain executive functions, input/output handling, bit manipulation functions, etc.; where the HOL cannot be used or is too inefficient to satisfy performance requirements. The percent of executable code which must have been generated directly from a HOL should be specified. It should be noted, however, that the real time requirements of TS may require use of machine language for certain functions. SWQA

should coordinate with the software development organization to assure that design standards are developed, published and met. SWQA should review the standards to assure that they are adequate, complete and monitor the design development activities to assure that they are available and are being used. All deviations should be reported to the design organization in writing for resolution.

5.2.2 Coding Standards

Coding from the design representation entails the conversion of that representation in accordance with the rules of the programming language used. Application of a set of standard rules is conducive to quick and systematic work. The resulting uniformity in the coding method is of particular advantage for maintenance of programs. The work of program coding has been in a state of transition for a number of years. Originally, programs could only be coded in machine language; nowadays the HOL is used nearly exclusively. This facilitates not only the work of coding itself, but also the communication between systems personnel. It must, however, be noted that some programming problems can best be solved by using assembly language.

SWQA personnel should assure that coding and programming standards are adequate and complete, are published, available and performed. SWQA should review the coding standards and their implementation to assure adherence to the following:

a. Coding is not started before the detail design representation is prepared and the programming specification is released; and a system for identification of the system elements and data objects is prepared.

b. A standard for the use of naming conventions and abbreviations for large labels and symbolic addresses is established. Names should be meaningful, reflecting both the function of the variable or label, and its structural relationship to other variables or labels.

c. Every code segment contains a single entry and a single exit (with the possible exceptions of routines where algorithm or data base is common to more than one extremely similar function, such as SIN/COS).

d. The beginning and end of any program or segment is completely contained in a single element.

e. Normally coding should be performed with one statement per source image line. In free format languages, if there is more than one statement per line, the statements must pertain to a common function.

f. Format statements, which are referenced in more than one place in a program, are grouped in one area to simplify debugging and maintenance.

g. Indentation of the source code is used to correspond to the logic structure of the program.

h. Data is grouped and arranged in a meaningful order.

i. Standardized formats are used for error messages. Cryptic error messages are prohibited.

j. Inhibit or rigorously control the conditions under which machine language patches may be used.

k. Every code segment is limited to reasonable amounts of logic, easy to understand, analyze, code and read.

l. Comments are used where necessary to enhance the readability and understanding of the program. Comments are

used frequently, sufficient to make the listing readable at a level higher than that of the code itself.

m. Loop variables are not altered during loop execution.

5.3 SOFTWARE VERIFICATION PLANNING

Software verification and test planning encompasses many elements of a software quality assurance program and essentially spans the total software life cycle. Software verification includes techniques that are used to assure that the computer program being developed will completely and correctly meet all of its requirements. It is the process of determining that the computer program was developed in accordance with the specifications, that it satisfactorily performs the functions for which it was designed and that it does not perform unintended functions. Software performance and design requirements which must be verified, include functional requirements, interface requirements and special (non-performance) requirements such as programming standards, program organization, protection of classified information, etc.

5.3.1 Verification Modes

Verification includes participation in reviews, studies and analysis, and designing and conducting tests to assure that critical requirements are being met. The goal is to improve the quality/reliability of the software and uncover problems early in the development effort so that they can be corrected at a much lower cost than if detected at a later period in the software life cycle. Verification is based on having a series of documents with a one-to-one correspondence, which defines and shows traceability between system requirements, software requirements, software design, software code and software tests. These documents include a system specification, a computer program product specification and a series of test documents - test plan, test procedures and test reports. Software verification is a continuous process of determing whether the output of each phase of software development meets all requirements imposed by the prior phase.

The requirements for "formal" verification of the software are specified in the QA section of the computer program development specification. This section will specify requirements for verification of the design and performance requirements contained in the requirements section of the specification. The verification methods may include inspection of the computer program, analysis of the computer program and demonstration or test of the computer program. The methods for verification of each requirement, with a one-to-one relationship between performance and design requirements and verification requirements, must be identified together with the success criteria.

The performance and design requirements should be stated in quantitative (verifiable) terms with tolerances where applicable. The methods of verification are defined below:

a. Test: Execution of the code under controlled conditions to generate and record realistic performance data. This data will be evaluated to ascertain compliance with requirements.

b. Analysis: Logical or mathematical processing of analytical or empirical data under defined conditions to show theoretical compliance with stated requirements. Analysis may include evaluation of internal control logic functions, numerical and statistical performance of algorithms and equations, sizing and timing parameters, core memory allocation, priorities, etc.

c. Demonstration: Execution of operational or functional capabilities before qualified witnesses. Instrumentation and

data recording normally will be provided indigenously by the elements being demonstrated.

d. Inspection: Examination or observation of the computer program against the applicable documentation to confirm compliance with specified requirements. Verification by inspection may consist of visual examination for the presence of desired characteristics or the absence of undesired characteristics.

A major portion of a test plan comprises a description of individual verification tests or demonstrations establishing the performance characteristics set forth or derived from the design requirements, as defined by the quality assurance provisions of the requirements specification and the external interface control document. In laying out the test plan, the originator should list all instances where detailed test procedures and test reports are to be submitted, whether customer approval is required, and the mechanics of the data transmittal/customer approval cycle. This is shown in Figure 5.3-1, which summarizes typical methods and required documentation for software verification. A requirements traceability cross-reference index is usually included in the test plan, though often it cannot be completed for the original release. This table correlates the quality assurance requirements with the specific test procedures that will verify each requirement. This tabular summary in an approved test plan document, followed up by approved test procedures and reports, constitutes evidence of compliance with contractual and quality assurance requirements necessary for product acceptance.

5.3.2 QA Review of Test Plans

Preliminary drafts of test plans should be available at PDR and completed drafts available at CDR. QA personnel should assist in the review and preparation of early drafts to insure clear and unambiguous plans are documented for software acceptance. As previously stated software should be treated like hardware from the standpoint that various levels of testing are required leading to ultimate acceptance. The major types of tests conducted on hardware Contract End Items (CEI), consist of bench level tests, qualification tests, and quality conformance tests (acceptance tests). Software testing is similarly structured with two significant differences that frequently cause confusion.

a. Quality conformance tests or acceptance tests applicable to hardware CEI's are not applicable in the same sense to CPCI's. These tests are conducted on each serialized unit of production line hardware to verify the integrity of the build process; that each unit has been fabricated per drawing and to verify workmanship. Such tests are not applicable to CPCI's because acceptance is based on different criteria. Once an operational program is coded, compiled and assembled into machine executable code residing on tape or disk, it's build process is complete and it is ready for validation. Once the machine executable code residing on a master tape or disk, etc., is validated there is no need to re-execute the validation procedure on duplicate copies. Verification of duplication is all that is necessary for acceptance of additional copies of the CPCI.

b. Hardware CEI acceptance is based upon successful completion of CEI qualification and acceptance tests. These are CEI level tests, not system tests. ATE and TS software, on the other hand, can only be validly accepted after system level tests. For example - an ATE software CPCI may consist of four major component programs - control, support, station self-test, and UUT test software. Proper execution of the UUT test program can only be verified in a total system environment requiring all four components functioning interactively.

TEST LEVEL	TEST CATEGORY	PROCEDURES CONTROL		ASSOCIATE VERIFICATION METHODS				DOCUMENTATION LEVEL	
		CUSTOMER REVIEW	CUSTOMER APPROVAL	INSPECTION	ANALYSIS	DEMONS	TEST	TEST PLAN COVERAGE	WRITTEN PROCEED
PROGRAMMER/LEAD ENGR DESK CHECK	DEVELOPMENT	NO	NO	X	X			NO	NO
PROGRAM UNIT DEBUG	DEVELOPMENT	NO	NO		X		X	NO	NO
PROGRAM UNIT INTEGR/ MODULE CHECKOUT	DEVELOPMENT	NO	NO		X		X	NO	NO
MODULE PERFORMANCE TEST	DEVELOPMENT	YES	NO			X	X	YES	YES
INTERMODULE COMPATI-BILITY TEST [1]	DEVELOPMENT [1]	YES	NO [1]			X	X	YES	YES
COMPUTER PROGRAM*(END ITEM) VERIFICATION [2]	FORMAL VERIF	YES	YES			X	X	YES	YES
SOFTWARE SUBSYSTEM VERIFICATION [3]	FORMAL VERIF	YES	YES		X	X	X	YES	YES
SW SUBSYSTEM/SYSTEM INTEGRATION & CHECKOUT	FORMAL VERIF	YES	YES [5]			X	X	NO	YES
SYSTEM PERFORMANCE DEOMNSTRATION	FORMAL VERIF	YES	YES			X	X	NO	YES
SITE INSTALLATION & ACCEPTANCE TEST [4]	FORMAL VERIF	YES	YES	X		X	X	NO	YES

[1] May include verification tasks defined as "preliminary qualification tests, which contain formal requirements.

[2] Also categorized as "formal qualification".

[3] May occur in conjunction with an at same site as software/system integration and checkout task.

[4] May be combined with or substituted for system performance demonstration task, depending on product end use.

[5] Informal system integration preceds formal verification of system/computing segment compatibility.

Figure 5.3-1. Typical Test Documentation Requirements Matrix

These differences and others affect the test planning process for ATE and TS software in several ways.

5.3.2.1 Validation. Ultimate software acceptance is essentially synonymous with system qualification. This implies that systems engineering is involved in, if not the prime source, for software validation test planning. Unless clearly addressed in the test plan, system specification requirements could be confused with CPCI Part I specification requirements as the origin of validation test planning documents and procedures.

5.3.2.2 Documentation. Acceptance of software is contingent upon completion of inspections, analysis, contractor verification and validation tests, demonstrations (i.e., fault detection demonstration for ATE) and ultimately the user conducted "hands on" tests. It is normally a systems engineering function to gather all such evidence of requirements satisfaction in support of eventual FCA, PCA activity. QA, as the acceptance agency of the contractor, has a vested interest in assuring this coordination of compliance data is gathered, maintained and organized in an official manner.

5.3.2.3 Evaluation. Test procedures are typically lengthy and complex requiring intimate familiarity with the system and programming language. Their efficient generation and use requires senior engineering personnel as test procedure writers and conductors. To validly assess the adequacy of such procedures as satisfying performance requirements, reviews must be conducted by personnel technically knowledgeable in ATE or TS system technology. Unless QA organizations have access to such personnel, this function should be delegated to a qualified independent test group.

5.3.2.4 Acceptance. Although CPCI's do not require acceptance tests for subsequent copies of the same program, the CPCI does require a full acceptance or validation test whenever it is used with a different processor configuration. This is frequently a consideration in ATE systems in which the same basic software controls different functional test stations. Since there can be slight differerences between processors/stations which affects software, acceptance tests should be demanded whenever the CPCI is transported for execution on a different station.

A good test program requires the development of a test plan, prepared in conjunction with the software requirements analysis phase, to define all levels of testing. For a complex computing system, several levels of testing are required to verify that all requirements are met:

a. Module, or computer program component testing will be conducted to verify that individual components have been properly coded and can be executed and that they satisfy corresponding software design and data base design requirements.

b. Intermodule testing is conducted to verify that software interfaces between computer program components and the executive program are satisfied.

c. Computer program testing will be conducted to verify that hardware/software interface requirements are properly implemented and software requirements contained in the computer program development specification are satisfied.

d. System testing will be conducted to demonstrate the operational system specifications.

All testing to determine compliance with the software specification requirements (computer program development specification) and the system specification requirements should be conducted by an organization separate from the software design group. Such independent testing is sometimes referred to as the "Product

Test" function. Product Test includes preparation of test plans, test procedures and test reports as well as the test cases themselves and the conduct of the tests. The only exception to this is the computer program component or module test to assure that each module satisfies its design requirement. This test is generally developed and conducted by the software designer to assure that each module implements the design contained in the draft computer program product specification. The other higher levels of test should be planned and conducted by an independent test or quality assurance organization that does not have a vested interest in the design. Their only concern should be the testing against requirements, both specification or baseline requirements and cascading requirements, such as those documented in a User Manual.

The QA representative should review the test plan for adequacy. A checklist, such as the following, should be prepared and used to assure satisfactory completion of required tasks.

a. The software products to be tested are identified.

b. The extent of testing is adequate and clearly defined.

NOTE

The problem of determing test sufficiency, as indicated by b. above, should be recognized during the test planning phase, i.e., it is not possible to review test plans/ procedures to ensure that all branches and logic will be exercised. Test planning should therefore include the use of tools such as analyzers as described in paragraph 5.4. These tools can be used during test execution to determine the branches and code exercised during execution of a test case.

c. All requirements of specification are covered.

d. Test support software and hardware is referenced and described.

e. The roles and responsibilities of each organization participating in software testing are clearly defined.

f. Methods of test control and status reporting are adequately described.

g. Test schedules and locations are specified.

h. Test cases are defined for all requirements, the test approach/method is appropriate, input/output data are adequately defined, success criteria is adequate.

i. The general procedures for analysis of test results are given: including identification of computer programs to be used for data reduction and/or analysis.

j. The test plan is prepared per the applicable documentation standards.

5.3.3 QA Reviews of Test Procedures

SWQA measures related to verification testing are primarily concerned with the adequacy of the test cases, the test plan and test procedures, the conformance of test conduct and test results with test procedures, and the timely reporting and correction of all software deficiencies. As a minimum, the test plan is reviewed in detail to ensure that the technical planning and test case definition is adequate and complete; that appropriate levels of testing are planned. The test plan should identify the schedules, test methods and success criteria as well as all required support facilities, equipment, software and personnel. It should include plans for testing at both nominal and extreme conditions. The individual test procedures are reviewed for adequacy, success criteria, callout

of required configurations of support hardware and software and the software under test. The test procedures are reviewed to assure that adequate detail is provided to clearly define the test objectives, test inputs, expected results, data recording requirements and data reduction requirements. Test procedures are reviewed against test (quality assurance or verification) requirements of the computer program development specification and the system specification to assure that all requirements are verified. Traceability between verification (by test) requirements and individual tests should be documented.

Formal program milestone reviews and Test Procedure Reviews (TPR) should be held for computer program verification and higher level tests to provide formal approval of the test procedures prior to the start of test. Individual test procedures should be reviewed to ensure that all significant design features will be tested and all specification requirements will be verified. The review will establish that there is compatibility between the test plan and the test procedures, test inputs are capable of testing the design at its limits, the test will satisfy all test objectives, test instructions are clear and concise, the configuration of the item under test is accurately described, all test hardware and software are aceptable and adequately defined, and the success criteria is consistent with the test objectives and clearly stated. These reviews are conducted by the test organization and attended by software design, software quality assurance, and other interested or participating organizations.

The QA representative should review the software test procedures for adequacy. A checklist, such as the following, should be prepared and used to assure satisfactory completion of the requirements.

a. Test procedure documents are prepared per the applicable documentation standards.

b. TPR's are scheduled and held prior to test conduct.

c. Test procedure documents are available as scheduled and their status is accurately reported.

d. Test procedures are consistent with the computer program test plan and are current with the design documentation.

e. There is traceability for each requirement to the test that verifies it.

f. All significant design features will be tested.

g. Test inputs and outputs are identified and success criteria is adequately defined.

h. The sequence of ordered steps to be executed is given for each test.

i. Test instructions are clear and concise.

j. The software configuration to be tested is described.

k. The hardware and test software configurations are identified and adequate for the test.

5.3.4 Verification Planning Summary.

The independent verification of software requirements should be accomplished by a SWQA function which is separate from the software development function. These two functions should be located in separate organizations within the project to provide an independent check and balance on the development effort. The technical basis and decisions upon which the design, code and test are founded must be challenged and feedback provided to the software development function to correct errors or deficiencies. The personnel assigned to perform these SWQA functions must possess technical skills in both the software engineering and QA disciplines.

Verification of requirements should be started early in the software life cycle, prior to preparation of the computer program development (requirements) specification. This allows time to review the system requirements and their allocation to computer programs and to plan the detailed software verification activities. The participation in software verification activities by software QA personnel provides the knowledge and data necessary for software acceptance and provides a higher level of confidence in the software quality/ reliability. This continual verification activity directly supports the formal software configuration audits (FCA/PCA) by assuring that the software documentation and computer program code are consistent, complete and satisfy contractual performance and design requirements. Software verification is a major element of a successful SWQA program. It is aimed at: assuring high quality/ reliability software products which satisfy contractual requirements; reducing the frequency of software errors; and reducing the overall software life cycle costs.

Section 6.0 summarizes the actual implementation of software verification and describes the conduct and controls of that activity.

5.4 TOOLS, TECHNOLOGIES, and METHODS

A number of tools, techniques and methodologies are available to assist in the development and assurance of high quality software. While their availability is limited at this time, eventually the number of these will expand considerably. Since most of the presently available tools are for the use of the software developers, the software quality engineer should be primarily concerned with evaluating the application of those selected, to assure they are properly used to bring about the intended improvements in software quality. In the future, when a diversity of such tools, techniques and methodologies are available, it will be necessary to narrow down the selection to those which will provide optimum results.

5.4.1 Application

Provisions for the use of tools and methodologies should be identifiefd in the projects' software planning documentation, such as the CPDP, management plan or quality assurance plan. It is not always possible to identify the specific tools at the time the planning is established; therefore, provisions for evaluating and using tools should be part of the initial planning. The specific selections and criteria for application can be defined later. This planning must include provisions for budgeting, staffing and using facilities.

The USAF software quality engineer should become familiar with the tools/ methods and their uses and coordinate with the software development and test organizations to participate in their selection and planning for application. He should assure that all tools are identified, controlled and documented, and are suitable for their intended functions. He will also provide surveillance over applications. Examples of tools and methodologies which have been developed, and may be used or modified for use, are discussed in the following paragraphs.

5.4.2 Type of Tools

Examples of the kinds of tools and techniques currently used in the industry to help assure software quality are briefly described below. USAF acquisition engineers should adequately familiarize them-

selves with these tools to determine if they warrant special attention in the RFP.

a. Review. These consist of reviews of the design and design implementation at various points. The "peer code review" is a currently used practice primarily associated with structured programming methods. It consists of evaluations by members of the coding team evaluating each team member's code for both correctness and assurance of good coding practices.

b. Analyzers. Analyzers are currently used to determine the degree of coverage by test plans/procedures and also to provide some metrics on the characteristics of the code and how it is used. Analyzers can be used to improve test coverage and, also to some extent, to detect deficiencies in the functional test and design rquirements. They also provide the programmer with data to assist in optimizing certain aspects of the implementation of the code.

c. Assertion Checkers. These allow the programmer to evaluate the ability of the program to detect illegal conditions. This is done by providing the programmer the means for inserting illegal (out of specification) conditions and observing the response.

d. HOL Debug Tools. These are software tools which assist the programmer in debugging code and in making the detection of errors more certain, easier and faster.

e. Top-down Programming. This technique minimizes the need for discrete stages of software integration testing by conducting the program design from the top-most specification requirements and integrating them with the levels immediately below. By this method, when the computer program has been fully coded, it also has been tested and integrated.

f. Proof-of-Correctness. This is a technique which provides for the application of assertions, to all logic paths in the computer program under evaluation, to positively establish that the functioning of the program cannot be wrong.

g. Statistical Predictions. These are methods by which estimates of the number of remaining, undetected errors can be predicted by evaluating the previous error history of the software during its development. Estimating techniques also include estimations of thoroughness of testing by the rate of detection of "simulated faults" in ATE software.

h. Auditors. Code auditors are tools which facilitate the evaluation of code against predetermined coding standards.

i. Configuration Audit Tools. These are methods used to assure the validity of the computer program configuration integrity by selectively tracing changes to the configuration of the computer program.

j. Automatic Test Program Generators. Programs which produce test cases, such as coded UUT modeled data - (circuit schematics and timing diagrams) producing an output pattern of UUT responses useful in testing and fault-isolating a UUT.

k. Automated Configuration Management Aids. Programs which provide automatic file cataloguing, configuration verification to assure valid test results, file protection against unauthorized change, and program comparators which detect differences in programs from the baseline.

l. Media Converters. Programs which provide automatic duplication, conversion and verification of data from one media (cards to mass store mass store to tape, disc) to another.

m. Automatic Tech Order Generators. Programs which accept and process file

listings of Abbreviated Test Language for all Systems (ATLAS) test programs, program boiler plate, UUT parameter file, UUT part number and produces a properly formatted USAF Tech Order.

n. Interactive Text Editors. A program maintenance tool which allows the user to construct and modify program source code or other textual information.

Many of the above automated developmental maintenance, and configuration control aids are available from the ATE or TS system contractor as a part of his basic operating system. Use of such tools/system features can reduce software development time significantly over manual methods. Contractor SWQA personnel should be sufficiently aware of a vendor's operating system capability in order to make judgements as to the validity of allowing use of such tools (such as media converters or text editors) as official configuration control or validation aids. The reader is referred to Section 7.0, Bibliography references 3 and 4 for a further discussion of software program developmental and QA tools.

Section 6.0 CODE & CHECKOUT, TEST & INTEGRATION, AND INSTALLATION PHASES

The activities discussed in this section cover the period from CDR through delivery. It is during these phases that the as-built software configuration is established, checked and validated. It establishes whether disciplined design and analysis phases have been completed; whether the requirements have been properly defined, interpreted and allocated to software; and whether requirement changes are minimal. If so, coding, debugging and validation efforts should be free of major redesign set-backs. Unfortunately, it is not realistic in software development, especially in simulation and test software to expect no changes. Technologies are too dynamic. Improvements are constantly being made to the weapon system performance, accuracies, etc. These changes correspondingly affect simulation and ATE software. Therefore, software developers and QA must plan for efficient change management. The guidebook on "Software Configuration Management" CDRL item A00E, addresses the totality of change management for ATE/TS software. This section is limited to a discussion of those QA activities necessary to verify software baselines and account for controlled changes. The four areas of QA activity which support the verification of software as-built configuration are:

a. Configuration verification & status accounting

b. Library and software media control

c. Control of formal testing

d. Formal configuration audits

The following paragraphs discuss QA conduct of, or support to, these functions.

6.1 CONFIGURATION VERIFICATION AND ACCOUNTING

The subject of configuration verification and accounting is used in this context as meaning all activities which support the verification of software configuration baselines as modified by controlled and approved changes. There are fundamentally two types of documentation required by QA to achieve configuration verification:

a. released engineer documentation

b. manufacturing planning and accounting documentation

The following sections describe how QA uses this documentation to verify software configuration.

6.1.1 Engineering Configuration Definition Documentation

The basic configuration description document for CPCI's is the Product Specification Part II per Mil Standard 483. It includes logic flows, listings and narrative descriptions of program operation. It is theoretically approved at CDR as the computer program product baseline. Other documents, however, are required to define the configuration of each version of a CPCI program as it progresses through formally controlled development and testing phases. The VDD is an exact description of each individual version of a CPCI and is used to keep track of detailed changes incorporated during program development. It is a modular index of subroutines comprising a given version and will list all functional program components, media, labeling requirements support software, utilities, validation and installation instructions necessary to build, accept, and load and

verify the machine executable programs (object code) into the designated target computer. It is equivalent to an Line Replaceable Unit (LRU) top assembly drawing.

Some contracts may employ equivalent documentation schemes such as the milestone system per SSD Exhibit 61-47B. This system substitutes 8 milestone documents which accomplish the intent of MIL-STD 483 Part 1, 2, requirements while providing a more granular documentation structure that accomodates better functional separation of responsibilities. In such a scheme the document equivalent to the VDD is the milestone 8.

6.1.1.1 Ancillary Software Products. Some ancillary software products are necessary in the development and acceptance of CPCI programs, such as:

a. Automatic Test Program Generators

b. Mission data programs

c. Memory load/dump routines

d. Patch loader programs

e. Test/simulation programs

f. Data reduction programs

g. Diagnostic programs

h. Vendor O/S software

These programs may be components of a deliverable CPCI or CPCI's themselves. Usually they are neither. Nevertheless, they are used in acceptance activities for deliverable CPCI's and must be configured and accounted for.

6.1.1.2 Documentation Accountability. Whichever documentation system applies, the following program features should be configured, verified and changes thereto accounted for as governed by program requirements and complexity.

a. A narrative program description

b. A logic program flow

c. A source code program listing

d. Applicable support software

e. An object code program

f. Acceptance or validation requirements

g. User instructions

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, drawing 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 period during which engineering defined configuration description documents must be formally controlled, typically begins with CDR, or completion of intermodule testing. This signals the completion of developmental testing during which engineering controls software configuration internally. As stipulated in the CPDP and CM plans, QA begins at this point. QA controlled tests are variously called Product Assurance tests, Contractor Internal Verification test, System Validation tests, or simply acceptance tests. Paragraph 6.4 discusses the attributes of formal QA controlled tests. At this point, however, let us examine the non-engineering documentation required to accomplish CPCI configuration verification and status accounting.

6.1.2 Manufacturing and QA Documentation

In addition to released engineering documentation, there are numerous manufacturing and QA documents and records which must be initiated to control and monitor software configuration. The USAF role here is to understand how contractor QA personnel assure that the as-tested configuration is controlled during testing, and how changes in program configuration are accounted for to insure the "as-delivered" program configuration matches the "as-approved." USAF personnel should periodically audit these contractor functions to insure objective records of program configuration accountability and test are available to support the submitted delivery documentation. The following describes the function of the major manufacturing and QA documents.

6.1.2.1 Test Planning Order. This order defines the step by step process by which the software product is to be built (i.e., assembled and compiled, and linked if applicable) tested, duplicated, verified, labeled and acceptance stamped. It defines in extremely precise detail the sequence of operations required to formally build, test and accept the applicable program.

A typical format for a software verification or planning order is shown in Figure 6.1-1. Although simplified this figure demonstrates the orderly fashion in which formal software testing is accomplished. Its objective is to provide a record which leaves no doubt as to exactly what was tested and how such tests were conducted. It assures all the necessary engineering drawings and test procedures are released authorizing what to test and how to test it. It assures all operations are completed satisfactorily before closure of the order and release of the software under test as an accepted product.

6.1.2.2 Discrepancy Report (DR). This report, sometimes called a rejection report, unplanned event report, material review report, pick-up, etc. is the contractor's official form for reporting product discrepancies. It is used to document a physical product deficiency, nonconformance, a procedural error, a planning error or any non planned event which impacts the product build-test-and-accept cycle. It must be satisfactorily dispositioned prior to closure of the test planning order. In software testing, it is used principally to document changes in program configuration such as machine language "patches" needed to incorporate make-work changes to the program. In this capacity, it serves as an interim configuration descriptor for the software, but since it is not officially released engineering, it must be held open until an engineering change is released to define the configuration departure made via the DR.

TEST PLANNING ORDER	TEST NO.		PART NO.
REQUIRED ENGINEERING DOCUMENTATION DWG NO. REV CHANGES TEST PRO. REV CHANGES			APPROVALS MFG _____ QA _____
EQUIPMENT REQUIRED	PART NO.	NAME:	REVISION: CHANGES:

OPERATION NO.	GROUP		COMPLETED	DISCREPANCY REPORT NO.
1.	QA	Verify satisfaction of test prerequisites per test procedure xxx-xxxxx	(Inspections stamp & date)	
2.	Mfg	Assemble test equip per xxx-xxxxx	"	
3.	QA	Verify operation 2.	"	
4.	Test	Load and verify tape ____ into ____	"	
5.	QA	Witness operation 4.	"	
6.	Test	Conduct validation testing per xxx-xxxxxx	"	
7.	QA	Witness operation 6. Stamp/ date each data page	"	
8.	QA	Verify closure of all test discrepancy reports	"	
9.	QA	Stamp (reel, cassette, disc) software.	"	

Figure 6.1-1. Formal Test Planning Order

6.1.2.3 Planning Accountability Record. A planning accountability record (PAR), or equivalent, is required to insure that all planning orders written to implement an engineering change are accounted for and completed in a given product. Software is no exception. For example, a given change might require:

a. Use of a new compiler

b. Merge of two CPCI's into one

c. Added functional capability

d. Installation into a new loading device

Accomplishing this change on a given CPCI might require 4 separate planning orders, one for each of the above functions. Separate orders are required because of concurrent activity at different locations. A PAR insures all work is completed prior to change closure.

6.1.2.4 Configuration Summary. This document is prepared by contractor QA from their completed planning order records. A separate configuration summary is prepared for each serialized end item to accurately describe change incorporation. It defines all the Class I and Class II changes incorporated in the product since its creation. For software, it is vital to insuring knowledge of configuration for a product embodying numerous changes.

The foregoing are the main non-engineering documents and records required by QA to maintain configuration status accounting. Figure 6.1-2 shows the sequence of events involved in using these documents to track software program configuration.

6.1.3 Verifying Program Configuration

Unlike hardware, which is physically inspectable, the job of verifying program configuration is quite different. It is neither practical nor necessary to physically inspect a machine language program. Rather, the functional configuration of a piece of computer sensible media is validated and then that media is identified and controlled. From the master tape (or disk), duplicates are made and verified using utilities designed for that purpose. Multiple copies of the master, plus hard copy listings, preserve the validated program configuration. If any changes (machine language patches) are needed to validate a given program, those patches must be defined by released engineering and configured on the VDD. In the hardware world, acceptance is based upon first inspecting the article to insure it has been built per print followed by acceptance tests to prove its functionality. For software, acceptance is based on functional tests supplemented by inspection and analysis of program listings. Since acceptance by analysis involves examination of a source program listing, flow and narrative for theoretically correct functioning, the assumption exists that the examined source code has been properly assembled compiled, link edited, etc. in accordance with functionally proven support software and documented procedures, such that, the machine executable program could be recreated from the source code and support software as defined by drawing. The ultimate objective of program configuration verification is to insure the as built and tested program conforms to the approved baseline plus approved, controlled changes. In paragraph 6.4 we shall examine some actual techniques for controlling software configuration during test.

6.2 LIBRARY CONTROLS

Computer Program Library (CPL) controls may be implemented in a variety of ways. They may be established and controlled by the SCM organization, QA or engineering. Although strongly supportive of configuration management, the requirement for library controls is specifically required by MIL-S-52779, and hence is

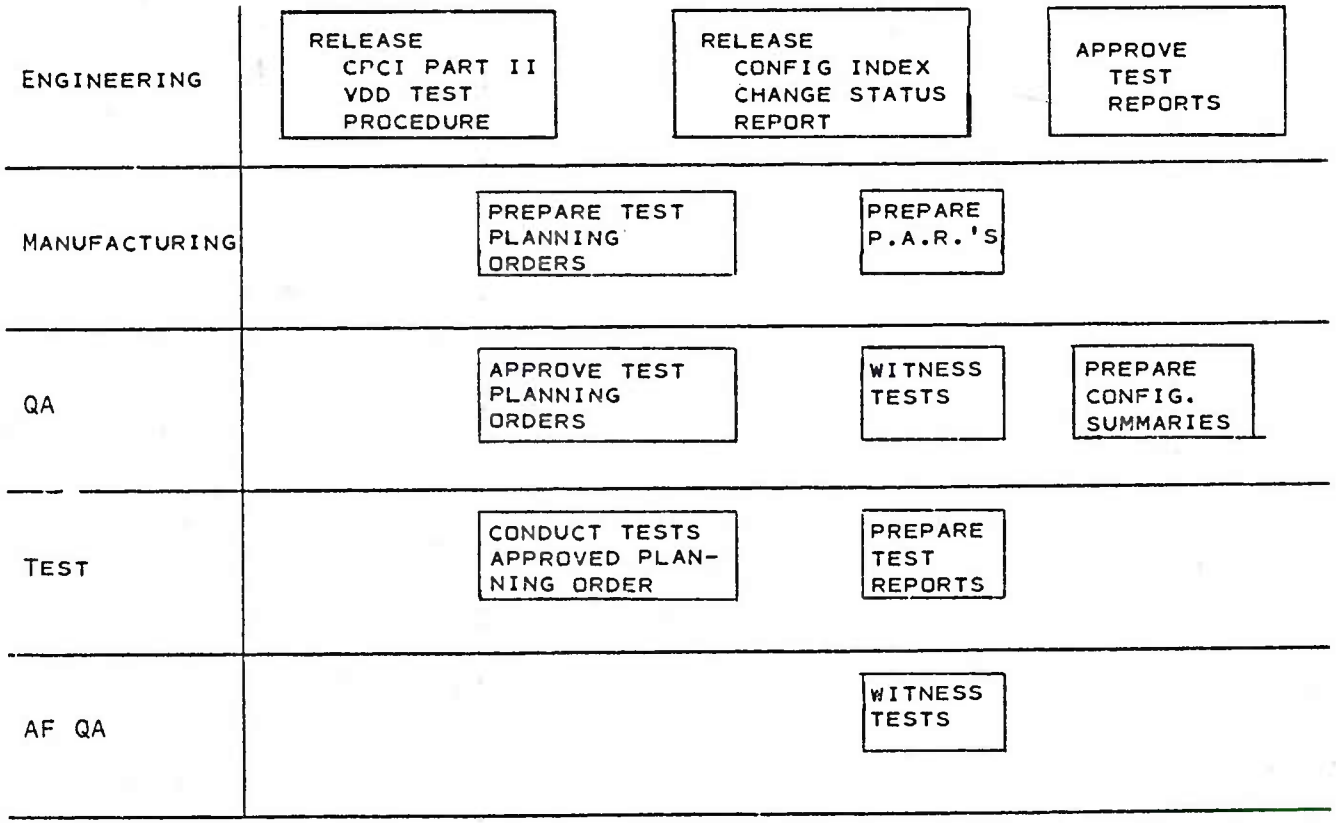

Figure 6.1-2. Formal Software Test Control Responsibilities

discussed in this guidebook. CPL controls are essential for the following reasons:

a. To assure that programs and data are properly identified

b. To prevent damage, degradation, loss or unauthorized change

c. To facilitate transfer of data to computer memory

A good CPL operation should provide for organized and protected storage and maintenance of:

a. Program documentation

b. Source and object program materials

c. Change records and authorizations

d. Problem reports and corrective action records

e. Support programs

The CPL should be readily accessible to the programming and test groups. The library function should be thoroughly documented and audited by QA. A detailed CPL procedure should be released and maintained current so that users can understand how their materials are organized in the library and what they must do to access such materials. Since one of the prime purposes of the library is to prevent unauthorized change to media or documentation, the CPL procedure must describe or reference the process by which previously approved source code lists, decks or object lists may be changed. The CPL function should be established to smoothly integrate the functions of media management, change management, problem reporting and status monitoring. Records and inventories should be maintained of all active and inactive program versions both validated and unvalidated. Let us now examine some of the details involved in operating a CPL.

6.2.1 Documentation Control

Formal released engineering documentation control is not a function of a CPL, but rather of a formal central Engineering Release Unit (ERU). CPCI listings (source and object) however, are not normally considered part of the CDR approved engineering baseline and therefore must be maintained under internal control (CPL control) until validated. This is the only practical way of controlling the configuration of program code. CPL controlled documentation includes such listings and other documentation (data base descriptions, load maps, etc.) needed to configure the applicable CPCI version for eventual formal release in the VDD. This sort of documentation must be maintained under strict control by the CPL. For example, consider a module of a TS CPCI. Upon completion of module verification tests by the programmer, the module is submitted to the CPL. At this point, the programmer can make no more changes to his program without authorization. This is to insure that other programmers who may be designing interfaces with his module can do so with predictable results. The source code for the module may still require modificaton after submission to the CPL, however it is done in a controlled fashion by obtaining review and approval by a designated authority. The change request and approval cycle for such a change is dealt with in depth in the Configuration Management guidebook. The only point being made here is that source code changes need to be controlled while not requiring formal ERU authority until the formal test phase. Management of such changes is a function of SCM internal change control and the mechanics of implementation is a CPL function.

6.2.2 Program Media Control

This subfunction of the CPL is concerned with organization and protection of the computer sensible media. Loss of data due to defective media can be disastrous. Included in the category of "media control" is:

a. Storage and protection of tapes, decks, discs etc.

b. Media conversion materials and procedures

c. Media duplication and verification procedures

d. Media identification and revision systems

e. Media certification (free of defects) methods.

The objective of media control is data retrievability and configuration control. Decks that are disorganized, unidentified, mutilated etc. cause serious data losses and program delays. By establishing a centralized CPL function, the programmer is relieved of the burdens of media control. Tapes which will not load because of skew, noise or other permanent tape defects can cause critical schedule slides. Data which has been unsuccessfully converted from one media to another is another source of aggravation. Much is available in the literature on the quality, care and handling of magnetic storage media (Bibliography references 7 through 9). The main environmental and handling considerations for computer program storage media are summarized below.

a. Temperature, humidity and air cleanliness data should be specified for the media storage facility.

b. Periodic inspections/audits should be conducted

c. Storage racks, bins, and containers should conform to accepted practices.

d. The Library should be a controlled access area.

e. Smoking should be prohibited.

f. Magnetic tape should be supported by the hub, not stacked horizontally.

g. A media sign-out file should be maintained for tape copies. The master should not be released after validation.

If the QA organization does not run the CPL operation, it must adopt some method of controlling the validated object program media. In some organizations, a separate protected area is designated for storage of "QA master" tapes which have been or are undergoing validation. This area is not part of the CPL, however it supplements the CPL function by creating a QA controlled area for "working copies" of validated media. This area is typically called a "QA Master media file" and is a repository for all QA controlled software, not only ATE or TS CPCI items, but avionics, test, instrumentation etc., type software. It is advisable to categorize software in the CPL and in the QA master media file in groups as follows:

Validated - Active
Validated - Inactive
Unvalidated - Active
Unvalidated - Inactive

Validated-active media are those which are authorized for usage in current tests or mission simulations. Unvalidated-active are those currently undergoing validation, and until testing is successfully completed, the master of the baseline media undergoing a test must be maintained. Unvalidated-inactive media sometimes results when a given tape version begins a test and then is superceded by a new compilation. If part of the test is completed with the original version, that version must be retained as part of the planning order records. This situation is extremely rare and highly undesirable because satisfaction of a given test with two or more different software versions creates the problem of having to identify version differences and prove why certain test functions completed with the original version need to be re-run with the new version. The objective of this file

is to retain all versions of software used, either partially or totally, during the conduct of a given formal test.

6.2.3 Change and Discrepancy Record Control

There are two basic sources of program changes - design changes and problems or deficiencies. A detailed discussion of change mechanics is reserved for the guidebook on "Configuration Management". The CPL function, however must provide for maintenance of the physical media and documentation change records. Any change processing system must require some formalized record for identifying, describing and authorizing changes. Forms designed for software internal change control are typically called a Design Change Request, Software Problem Report (SPR) or variation thereof. However designed or named, these forms must be numbered, dated and contain the following information:

a. Name of originator

b. Reason for and class of change

c. Problem description

d. Proposed solution

e. Type of problem - coding, design, data base, computational logic

f. Versions and documentation updated

g. Organization/person responsible for fix

h. Approval authority

i. Version fix/change incorporated in

j. Retest requirements

The CPL must provide for retention and organization of whatever materials describe these change parameters.

6.2.4 Support Software Control

In order to be able to trace the origin of any given program version to its source materials, all of the support materials needed to produce a machine executable load module must be identified and maintained in the CPL. This includes:

a. Compilers

b. Assemblers

c. Link editors

d. Loaders

e. Code generators

f. Job control language programs

g. Various other support utilities

These needed support programs must be identified in engineering documentation, usually the VDD, to each specific CPCI version and the materials themselves maintained in the CPL.

6.2.5 Summary of Library Controls

The CPL function, therefore provides for all materials needed to create and maintain programs throughout the software development life cycle. Usually CPL controls will have been established early in the Full-Scale Development Phase of the prime weapon system and will include all weapon system software. By the time the ground systems are ready to require CPL controls, the existing CPL may already contain:

a. Avionics software

b. Special Laboratory Test Software

c. Data reduction and processing programs

d. Programmable Read Only Memory (PROM) software

e. Avionics simulation and support programs

The librarian's job therefore becomes increasingly more complex as ATE/TS programs are added to his facility. Careful planning, vigilant maintenance and QA auditing of the CPL function helps insure success of the software configuration control effort.

6.3 PROBLEM REPORTING AND CORRECTIVE ACTION

Software errors are a diverse and difficult lot. They range from the trivial (e.g. syntactic errors resulting from mis-use of a language, detected by the language processor) to the complex (e.g., of a logic error resulting only from the execution of a peculiar combination of program logic paths, detectable only by constructing an input case which forces the execution of that combination).

Software errors emerge from all phases of the software life cycle. Recent data tends to show a preponderance of errors resulting from the design phase. This may be because design is really the most error-prone phase of the life cycle, or it may be because most error reporting systems do not begin tracking errors until the software is placed under configuration control; a point in the cycle when most coding errors have been detected and corrected. Be that as it may, the source of errors has as diverse and difficult a picture as their nature. The detection of errors results from a software development process and a software quality process. Errors are also detected by the software user. The software quality engineer, for example, may detect errors through reviews and audits; through his role in the product test function; through the use of error-detecting tools whose use he has promoted and monitored; and through the formal testing process he helped define and execute. Because of the diversity of errors, their sources, and their ways of detection, a central control of error reporting system is necessary. The SWQA engineers should either motivate, enable or actually operate this centrally-controlled system.

According to a recent Boeing study, (Bibliography ref. 10) on a medium sized project, the primary sources of errors are logic, such as missing or incorrect logic (31.2%); and data handling, such as the mis-use of data, indices or flags (13.4%). No other single source accounted for more than 8% of errors encountered. (Categories, 22 in number were provided by RADC). If all data-related errors were totaled (data handling, data base interface and preset data base + global variable/compool definition), the result is 19.8%. Interestingly, interface errors were not significant (3.9%). The results of this study are shown in Figure 6.3-I. While this data represents that of an airborne avionics system software project, the generic causes of problems are universally applicable.

6.3.1 Problem Reporting Methods and Concepts

It is important to report and track software problems. It is also important to know when to track, and for how long. Typically in the software life cycle, a great amount of error detection will occur before a formal tracking system is necessary. Errors discovered in the analysis and design phases, for example, are generally not tracked since they are usually corrected in the emerging specification and design as they are discovered. Even during the code and check-out phase, errors usually are not tracked because they are too numerous and because the time to fix is often less than the time to report. It is after a software product begins acceptance testing, or is released to the customer, or begins system integration (preferably, whichever comes first), that a formal tracking system becomes necessary.

The purpose of an error tracking system is:

	CATEGORY	NUMBER	PERCENT %
A	COMPUTATION	109	5.4
B	LOGIC	635	31.2
C	I/O	28	1.4
D	DATA HANDLING	272	13.4
E	OS/SYS. SUP. S/W	8	0.4
F	CONFIGURATION	12	0.6
G	ROUTINE/ROUTINE INTERFACE	41	2.0
H	ROUTINE/SYS. S/W INTERFACE	3	0.2
I	TAPE PROCESSING INTERFACE	6	0.3
J	USER INTERFACE	12	0.6
K	DATABASE INTERFACE	17	0.8
L	USER REQUESTED CHANGES	161	7.9
M	PRESET DATA BASE	67	3.3
N	GLOBAL VARIABLE/COMPOOL DEF	48	2.3
P	RECURRENT	148	7.3
Q	DOCUMENTATION	27	1.3
R	REQUIREMENTS COMPLIANCE	144	7.1
S	UNIDENTIFIED	30	1.5
T	OPERATOR	158	7.8
U	QUESTIONS	19	0.9
V	HARDWARE	32	1.6
X	NON-REPRODUCIBLE	62	3.1

Figure 6.3-1. Software Errors

a. To ensure that errors are corrected, not forgotten.

b. To ensure that all error corrections are approved before changes are made.

c. To enable the measuring of error correction progress.

d. To provide feedback to the user on error status.

e. To prioritize the order in which errors are corrected.

The role of SWQA, in this discrepancy reporting and corrective action process, should include some or all of the following-definition of the system, including design of the necessary forms and reporting methodologies.

a. Tracking open error reports, to ensure that none are forgotten and that high priority problems are worked first.

b. Participating in change reviews, to represent the quality viewpoint in the decision-making process.

c. Ensuring that error status is reported to the software users (if any).

d. Auditing correction progress to be sure that the software is getting better and not worse (e.g., that the number of uncorrected errors is decreasing).

6.3.2 Internal Errors

All errors reported under the discrepancy reporting system should be recorded on a Software Problem Report (SPR) form. The SPR becomes the primary vehicle for tracking problems. A sample SPR form is shown in Figure 6.3-2. The process for processing this form is discussed in the succeeding paragraphs of this section.

6.3.3 External Errors

If an error is detected during a formal QA controlled test, it is regarded as official test discrepancy (external to the software development organization), subject to formal discrepancy reporting corrective action mechanisms. These formal mechanisms are costly to process since they usually require use of a corporate level reporting form; copies of which are sent to central discrepancies accounting organizations. These types of forms usually require a statement of corrective action prior to closure since they assume the discrepancy being reported is a true deficiency which should not have occurred. Daily, routine software development errors which, from a practical standpoint are unavoidable, should theoretically not occur during a formal acceptance test, and hence not be reported on corporate discrepancy tracking paperwork. Care must therefore be taken in early stages of CPDP preparation, not to submit software to formal test controls prematurely.

6.3.4 Problem Report Processing

USAF personnel charged with acquisition or subsequent auditing should ensure that the contractor implements some disciplined method of problem report processing. Problem report processing should entail:

a. A standard serialized reporting form

b. Analysis for impact

c. Categorization for trend analysis

(1) self contained-vs-interface error
(2) documentation error
(3) enhancement change
(4) test case/procedure error

d. Authorized approval

e. Logging and monitoring for trends

f. Corrective actions

g. Authorized closure

h. Adequate retest and engineering evaluation

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)
- ☐ HDL Debug
- ☐ Analyzer
- ☐ Simulation
- ☐ Assertion
- ☐ Proof
- ☐ Other

Description of symptoms _____

Configuration level _____

Correction importance/need date _____

Authorizing Signature _____ Drgn. _____ 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 _____ Drgn. _____ Date _____

Figure 6.3-2. Suggested Software Problem Report

i. Adequate engineering change coverage

6.3.5 Problem Reporting Summary

It must be emphasized that although all software errors are undesirable, those encountered during a formal test are most expensive. Designers frequently claim that there is a point of diminishing returns reached in developmental testing; that no matter how much informal testing is done, additional errors will inevitably arise from exercising the software in a system environment. This is certainly true, and since schedules cannot accomodate a so-called "dry run" (non-QC witnessed) of a entire system level validation test, some system level software problems will have to be handled formally.

6.4 VERIFICATION AND VALIDATION TESTING CONTROLS

In paragraph 5.3 we explored the planning aspects for verifying that software meets its requirements. We identified the ways in which requirements compliance can be demonstrated, what to consider in planning software tests and what to look for in reviewing test plans and procedures. In the following paragraphs, we shall discuss the controls which must be exercised by QA in actually conducting or monitoring the conduct of formal tests.

6.4.1 Attributes of a Formal Test

As discussed earlier, the contractor performs several levels of testing in the development of CPCI's. Module verification tests and module compatibility tests are usually regarded as engineering tests or confidence builders that the design is proceeding correctly. They do not sell off or acceptance test anything. Formal tests on the other hand, are any tests designed specifically to provide objective evidence of satisfaction of released engineering requirements. Formal tests must be conducted under the auspices of QA and are characterized by the following controls.

a. The article under test must be formally configured.

b. The test requirements must be defined in released engineering.

c. The test procedure must be released engineering.

d. Test support equipment must be formally configured, acceptance tested or calibrated/certified as required.

e. A formal test planning order is required (ref. Fig. 6.1-1) .

f. All test prerequisites are defined and completed.

g. QA witnessing or monitoring of test conduct is required.

h. Test discrepancies are formally documented.

i. Test procedure changes require engineering approval.

j. Formal test reports are prepared.

k. UUT or equipment configuration changes are formally released.

l. Test set-up must be formally defined.

m. Completed records contain "as run" version of test procedure.

n. Test planning order contains a complete history of test events, planned and unplanned.

o. Formal reviews are held prior to and after test.

p. QA stamps and signs the UUT and test planning order.

Software tests designed to demonstrate compliance with CPCI Part I or equivalent documentation, are considered formal tests requiring controls equivalent to the above. The sequence of conducting formal tests is as follows:

Release Test Plan Test Readiness Review (TPR) Pre-Test Briefing (PTB) Test Conduct Issue Test Report	USAF Review and Support

QA should participate in all of the above and must play a major role in test conduct. The following paragraphs discuss the main events involved in formal testing.

6.4.2 Test Readiness Review (TRR)

The TRR is a formal event held approximately two weeks prior to test start with USAF in attendance. All functional organizations are required to attend to review the test planning status, identify open items and assign action items. A typical TRR agenda includes.

a. Statement of Test Objectives, Requirements

b. Statement of Success Criteria

c. Identification of Equipment, Test Conductors, Procedures

d. Status of Open Items Prerequisites

e. Description of Test

f. Facilities, schedules

g. Documentation Releases - Drawings, Planning Orders, etc.

h. Assignment of Action Items - due dates

The TRR insures all concerned organizations have the opportunity to voice objections, provide corrections and obtain a clear understanding of the test as it is planned. These reviews are absolutely essential for a complex test program. The TRR is usually chaired by the designated lead test conductor and is attended by levels of management from software design, systems engineering, QA manufacturing, SWQA and equipment design; who are authorized to commit for their organizations. When all are satisfied that their concerns and open items are identified and action items assigned, the meeting is adjourned. The chairman follows up with minutes documenting the action items.

6.4.3 Pre-Test Briefing (PTB)

The PTB is held to assure that open items identified at the TRR have been acceptably resolved and to declare "all is go" for the start of tests. Some of the QA concerns to be registered at these reviews should be:

a. Is the specified CPCI version to be tested, formally configured and released?

b. Are any machine language "hand-loads or patches" required, configured and released?

c. Are all known test procedures discrepancies resolved?

d. Is all required test support equipment properly calibrated or certified?

e. Are sufficient QA personnel available to support planned schedules?

f. Is traceability established from the CPCI test requirements to the procedures?

g. Is the test planning order released and approved?

h. Is all required test support equipment authorized?

i. Has the test line configuration been verified?

j. Has the customer been notified of test schedules?

k. Is the particular test being planned compatible with the master software verification plan?

Most contractor project QA organizations draw support from central QA organizations - factory or line QA, metrology services, QA engineering, etc. Whatever the organization, QA should be adequately represented at these reviews. Line QA and the project SWQA engineer should support these reviews to adequately address all of the above concerns.

6.4.4 Test Conduct

Usually the actual "sell-off" run of the test will be preceded by a so-called "dry run." This dry run is conducted by the test conductor to insure its sell-off run will be smooth, clean and free of discrepancies requiring costly but necessary discrepancy handling mechanisms. It also eliminates the psychological hazzard of conducting a formal test for QA and the customer, which is plagued by numerous failures. Unfortunately, in testing complex software systems, schedules oftentimes do not permit dry run testing. In ATE and TS software testing, QA should expect software discrepancies to occur, frequently requiring changes and appropriate retest. Therefore, QA must be extremely vigilant during the monitorship of testing. Unless discrepancies are meticulously documented, software configuration control can easily be lost.

The first question which must be answered is - should QA witness 100% every step of the procedure or will a periodic monitorship with spot-check witnessing suffice? This can only be answered by evaluating the particular situation at hand -

a. How experienced is the test conductor?

b. How error-prone is the procedure?

c. Are numerous recordings, observations required?

d. Is the procedure laborious and tiresome?

e. Is test support equipment subject to frequenty failure?

f. Is the CPCI subject to frequenty failure?

Whatever mode of monitorship is selected by QA, it should be justified and specifically called out by the test planning order - "100% witness" or "verify completion of ---." Next, QA must verify test set-up. Is the test configuration assembled, equipment calibrated, cables properly hooked up, etc?

The next operation will probably call for loading the CPCI program into the memory. This operation should be witnessed to insure the authorized software plus any applicable handloads have been successfully loaded. A common technique for verifying successful load is the "checksum verification"; an automatic comparison of a bit count of loaded data with a pre-calculated and stored checksum. This is usually required as part of the procedure and is "bought off" by the inspector. Once the baseline program is loaded, testing begins and inspection authenticates that the procedure was run as written. In software testing, frequent actual and apparent failures are encountered. All such occurrences are to be documented in the procedure at the discrepant step. Anomalies fall into four or five basic categories:

a. Hardware failure

b. Software failure

c. Procedural discrepancy

d. Operator error

e. Peripheral interference

Oftentimes, in software testing, peripheral interference can cause aggravating delays while, in fact nothing is discrepant. Radiated or conducted peripheral line transients (spikes on power lines, on-off cycling of heater

switches, static discharges, etc.) have been proven to cause apparent test failures. These are minimized by properly designing the test environment - using line filters, isolation transformers, screened rooms, etc. QA, however, must assume the worst, until it is adequately proven to be such interference. Upon encountering any discrepant condition during the test, the first step should be to record the event, even if it is uncertain as to cause. Usually the contractor's normal test conduct procedures, QA manuals etc., define how to process test discrepancies. Usually they are adequate for handling software test discrepancies as well as hardware, however, certain questions unique to software should be considered:

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

b. What was the fix? - machine language patch-or-newly recompiled program?

c. What locations and data were modified? Where is this information documented?

d. What controls insure only autorized memory locations were modified to fix a software problem after initial load?

e. What retest is required?

(1) for the portion of program failing?

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

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

g. Who authorized the fix and test restart?

h. Has all impacted released documentation been changed? - (Part II spec, VDD, Config. Index, etc.)

The above questions present an idea of the main concerns which must be exercised by QA personnel responsible for monitoring test conduct. Each test situation must be individually evaluated. To be effective, QA people should be vigilant and inquisitive at all times. They should ultimately be able to show, from the completed test planning order records, exactly what configuration of operational software resided in memory during each test segment, what sources of test data (real time or batch) were used during the test, what anomalies were encountered, how were they resolved, and what the final product configuration is.

6.4.5 Test Reports

The final QA activity relating to verification and validation testing is the review of the test report. The software QA engineer should review the test report for the following:

a. Have the planned objectives been satisfied?

b. Does the report discuss the degree of requirement satisfaction achieved by the test results?

c. Are unsatisfied objectives addressed in terms of a reschedule test committment?

d. Is there a complete description of the test environment?

(1) personnel conducting and witnessing the test

(2) output generated

(3) input data

(4) discrepancies and anomalies encountered

(5) equipment configuration used for test

e. Is the test report approved by authors of the test requirements?

6.4.6 Test Control Summary

The most important point to remember in QA control of testing is that records must result showing exactly what was tested to what procedures in satisfaction of what requirements. Test records, in conjunction with analysis and inspection records, form a total software verification package. QA must oversee the preparation and correctness of this total compliance package to be used in support of formal audits as discussed in the next paragraph.

6.5 CONFIGURATION AUDITS

Configuration Audits are conducted by customer personnel at fixed points during the system acquisition process. As defined by the CM plan, a Functional Configuration Audit (FCA) is normally scheduled sometime around the completion of CPCI qualification. A Physical Configuration Audit (PCA) is normally scheduled after fabrication and acceptance of the first article. For an ATE system, for example, the contractor will require an FCA/PCA on the vendor supplied portion of the ATE system (basic processor, test station and operating system software) prior to delivery to him. The Air Force, in turn will require and FCA/PCA on the entire ATE system - (processor, station, OS, adapter and UUT test program). The requirements for configuration audits are specified in MIL-STD-1521A and are addressed in depth in the guidebook on Reviews and Audits. The SWQA plan should include a requirement for retaining records needed to support formal audits. As discussed previously, the primary form of objective records used in support of FCA/PCA will be test records. Depending upon provisions of the contract, the PCA or first article delivery will determine the product baseline against which formal Class I changes must be processed, and incorporation thereof verified by QA. QA records must be available to support the objectives as outlined below:

6.5.1 FCA Provisions

a. Verification that CPCI qualification requirements including test plan, design, test procedures and results are properly documented.

b. Verification that the CPCI has been subjected to all specified tests which verify its performance.

c. Verification that the CPCI performs as required by its performance specification.

d. Verification that all approved changes have been incorporated and successfully tested.

e. Verification that test documentation is current and accurate with respect to the configuration of the CPCI.

6.5.2 PCA Provisions

a. All media, listings and documentation comprising the deliverable CPCI are properly identified.

b. The CPCI design conforms to applicable documentation and programming standards.

c. The end item subjected to PCA Is the same as that subjected to FCA.

d. QA and CM records adequately account for all approved changes. The types of QA/CM records and documents which support these verification objectives are:

(1) Completed test planning orders

(2) Planning accountability records

(3) Configuration summaries

(4) Discrepancy reports (Unplanned Events)

(5) Configuration Index (DI-E-3122)

(6) Change Status Lists (DI-E-3123)

(7) Version Description Document (DI-E-3121)

(8) Test Requirements Document

(9) CPCI Part I and II specifications

(10) Programming standards and conventions

(11) Test Reports DI-T-3717

(12) User's Manual DI-3410

(13) Test Plans/Procedures DI-T-3703

6.5.3 FCA/PCA Considerations

In order to prepare for a successful FCA and PCA, it is important that CPCI documentation updates keep pace with the progressing program code. Machine language patches or recompilations of new source code to produce corrected or more efficient code, should be accompanied by corresponding updates to the CPCI product spec, VDD, ICD, TRD and other documentation impacted by program configuration changes. Unless this is rigorously enforced, design documentation changes are usually put off for update "when time permits." This results in program code incompatibilities with the design logic narrative and flow when audited at PCA. Adequate preparation for FCA/PCA, therefore begins at CDR, when design begins its translation in code. A successful PCA signals the completion of the software development process. The CPCI is accepted by the customer and deployed as an operational component of Operational Support Equipment, (OSE). All applicable documentation is now under Class I control. The utility provided to the customer by the CPCI will in a large measure be a reflection of the effectiveness of the SWQA program.

Section 7.0 BIBLIOGRAPHY

1. Cost of Developing Large Scale Software, Ray W. Wolverton, IEE Transactivities on Computers, Vol. C-23, No. 6, June 74.

2. Seminar - Software Management, Phase III, L.A. January 26, 27, EIA-G33 committee actions.

3. D180-19846-1 Software Quality Assurance Handbook, Rev B, December 23, 1977, The Boeing Company.

4. NAVMATINST 3960.9, 9 Sept 76, Acquisition Planning Guide for Automatic Test, Monitoring and Diagnostic Systems and Equip.

5. Automatic Test Program Generation; Presentation to NSIA/AIA Meeting, 20 May 1975 by F. Kiguori, Naval Air Engineering Center.

6. Automatic Test Equipment Software Configuration Management; Dennis L. Wood, Software Enterprises Corp., Canoga Park, CA; ASSC record 72, p 110.

7. Long Term Retention of Data, EDP Analyzer, Canning Publications, Jul 73, Vol II # 7.

8. Cut Expenses by Taking Care of Your Tape; Robert G. Devitl, 3M, Minn. Minn Computer Decisions, Oct 1970.

9. Magnetic Tape Qualification & Acceptance Testing, T.J. Dylewski and R. H. Spitler, Inform. Processing, Lockheed Missiles & Space Co., Inc. Sunnyvale, CA.

10. "Software Error Data Acquisition", RADC-TR-77-130, April 77, Fries.

11. "Some Experience with Automated Aids to the Design of Large Scale Reliable Software", IEEE Transactions on Software Engineering, March 75, Boehm, McLean, Urfrig.

12. "Flowchart Symbols and their Use in Information Processing", X3.5-1970 American National Standards Institute, 1971

Section 8.0 MATRIX: GUIDEBOOK TOPICS VS GOVERNMENT DOCUMENTS

Figure 8.0-1 is a cross reference matrix showing the guidebook topics and government documents which address each topic. Elements in the matrix indicate guidebook sections in which the topic is discussed.

TOPIC \ DOCUMENT	AFR-800-14 VOL II	MIL-S-52779(1)	MIL-Q-9858	MIL-STD 1521A	MIL-STD 483	SSD EXHIBIT 61-47B	DOD DIR 500 0.29	MIL-STD 1520	AFR-65-3	MIL-STD 490	MIL-D-83468	MIL-STD 1519
SW QA PLANNING	2.8 3.9	3.2.1	●								SEC 4.C	
DESIGN REVIEW		3.2.6		APP B-D								
EST SW QA		3.2.1										
PREP. SW QA PLAN		3.2.1										
SOFTWARE STANDARDS		3.2.6										
SW VERIFICATION		3.2.3										
SW DOCUMENTATION		3.2.7			APP VI	●				APP VI APP XIII		●
CONFIG MGT	3-9	3.2.2			●		V		●			
CONFIG ACCOUNTING		3.2.2										
PROBLEM REPORTING		3.2.4										
CORRECTIVE ACTION		3.2.4						●				
REVIEWS & AUDITS		3.2.8		●								
TOOLS, TECHNOLOGIES		3.2.9										
LIBRARY CONTROLS		3.2.5										
SW TESTING		3.2.3									4.2 4.4	

Figure 8.0-1. Guidebook Topics Versus Government Documentation

Section 9.0 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.

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 - A series of instructions or statements in a form acceptable to computer equipment, designed to cause the execution of an operation or series of operations. Computer programs include such items as operating systems, utility programs, and maintenance/diagnostic programs. They also include applications programs such as payroll, inventory, control, operational flight, strategic, tactical, automatic test, crew simulator and engineering analysis programs. Computer programs may be either machine dependent or machine independent, and may be general purpose in nature or be designed to satisfy the requirements of a specialized process of a particular user.

Computer Program Development Cycle - The computer program development cycle consists of six phases: analysis, design, coding and checkout, test and integration, installation, and operation and support. The cycle may span more than one system acquisition life cycle phase or may be contained in any one phase. (AFR 800-14, Volume II)

Computer Program Configuration Items - A computer program or aggregate of 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 suitble for insertion in a digital computer.

Configuration Item - An aggregation which satisfies an end use function and is designated for configuration management.

Configuration Management - A management 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.

External Interface - Data passed between two or more computer programs or between a computer program and peripheral devices external to the computer in which the program resides. The data may

be in the form of an interrupt signal or may be a digital data stream either output from the computer or input into the computer for procressing.

Hierarchial Decomposition - A process which successively divides some entity (requirement, computer process, data element, etc.) into components progressively exposing more detail. Each decomposition is a complete restatement of the parent element (and only the parent element). The collection of all elements at any one level is a functionally complete entity.

HIPO Diagrams (Hierarchy plus Input Process Output) - A graphic design technique used to show functions in terms of the input to a process, the process itself, and the output results from that process. A typical HIPO package contains three different items; a table of contents, overview diagrams, and detail diagrams.

Internal Interfaces - Data passed between elements of a computer program and usually included in the computer program data base.

Logic Flow - A diagrammatic representation of the logic sequence for a computer program. Logic flows may take the form of the traditional flow charts or in some other form such as a program design language.

Naming Convention - Standard conventions applied to the naming of variables, data base items, modules and lower level computer program components. Such conventions should allow distinguishing a computer program components place in the hierarchial structure, by its common data base names from local variable names, and common subroutines from module unique subroutines.

Organic - A term used to designate a task performed by the Air Force rather than a contractor.

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.

Program Design Language - An English-like, specially formatted, textual language describing the control structure, logic structure, and general organization of a computer program. Essential features of a program design language are:

a. It is an English-like representation of a computer procedure that is easy to read and comprehend.

b. It is structured in the sense that it utilizes the structured programming control structures and indentation to show nested logic.

c. It uses full words or phrases rather than the graphic symbols used in flow charts and decision tables.

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

Software Quality - The primary characteristic of software quality is that the software performs as intended. This implies not only that the software reflects the specification to which it is written, but also that the software specifications themselves adequately address the system/mission requirements. Key attributes of software quality include: reliability, flexibility, traceability, testability, integrity, maintainability, and completeness. Quality software is: well-defined, well-documented, free of design deficiencies and coding errors, satisfies performance requirements, and has minimum life cycle cost.

Software Quality Assurance - A planned and systematic pattern of all software related actions necessary to provide adequate confidence that Computer Program Configuration Items (CPCI's) or other software products conform to established technical requirements and that they achieve satisfactory results.

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 Engineering - The application of scientific and engineering efforts to transform an operational need or statement of deficiency into a description of systems reuqirements and a preferred system configuration that has been optimized from a life cycle viewpoint. The process has three principal elements: functional analysis, synthesis, and trade studies or cost-effectiveness optimization.

System Generation - The process of producing a computer program from two or more program elements such that the separate program elements will perform together as an integrated whole.

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

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

Top Down Integration - The development, testing and integration of the separate structural elements of a computer program in a hierarchial manner beginning with the highest levels and working down through the lowest levels until the computer program is complete.

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 - Computer program validation is the test and evaluation of the complete computer program aimed at ensuring compliance with the performance and design criteria. 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 requirements levied by the previous step, including those set for quality.

Section 10.0 ABBREVIATIONS AND ACRONYMS

ATE	Automatic Test Equipment
ATLAS	Abbreviated Test Language for All Systems
ATPG	Automatic Test Program Generator
CCP	Contract Change Proposal
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CEI	Contract End Item
CM	Configuration Management
CPCI	Computer Program Configuration Item
CPDP	Computer Program Development Plan
CPL	Computer Program Library
DID	Data Item Description
DR	Discrepancy Report
ECP	Engineering Change Proposal
ERU	Engineering Release Unit
FCA	Functional Configuration Audit
HIPO	Hierarchial-Input-Process-Output
HOL	Higher Order Language
ICD	Interface Control Drawing
LRU	Line Replaceable Unit
OS	Operating System
OSE	Operational Support Equipment
PAR	Planning Accountability Record
PCA	Physical Configuration Audit
PDL	Program Design Language
PDR	Preliminary Design Review
PO	Project Office (AF)
PROM	Programmable Read Only Memory
PTB	Pre-test Briefing
QA	Quality Assurance
RFP	Request for Proposal
ROM	Read Only Memory
SAE	Software Acquisition Engineering
SCM	Software Configuration Management
SDR	System Design Review
SOW	Statement of Work
SPR	Software Problem Report
SRR	System Requirements Review
SWQA	Software Quality Assurance
TPR	Test Procedure Review
TRD	Test Requirements Document
TRR	Test Readiness Review
TS	Training Simulator
UUT	Unit Under Test
VDD	Version Description Document

Section 11.0 SUBJECT INDEX

*U.S.Government Printing Office: 1980 — 657-084/527