AD-A048 577 SYSTEM SOFTWARE ...TA MONICA CALIF F/G 9/2
...AGEMENT GUIDEBOOK: VERIFICATION.(U)
AUG 77 H BR. ... FINFER F19628-76-C-0236
UNCLASSIFIED SDC-TM-57... .../02 ESD-TR-77-263 NL

1 OF 1
ADA048577

END
DATE FILMED
2-78
DDC

ESD-TR-77-263

SOFTWARE ACQUISITION MANAGEMENT GUIDEBOOK: VERIFICATION

(2)

System Development Corporation
2500 Colorado Avenue
Santa Monica, CA 90406

August 1977

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the 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 conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

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

William J. White

WILLIAM J. WHITE, Capt, USAF
Project Engineer

John C. Mott-Smith

JOHN C. MOTT-SMITH
Project Leader

John T. Holland

JOHN T. HOLLAND, Colonel, USAF
Chief, Techniques Engineering Division

FOR THE COMMANDER

Toru Yamamoto

TORU YAMAMOTO, Colonel, USAF
Director, Computer Systems Engineering

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(18) 1. REPORT NUMBER: ESD-TR-77-263

2. GOVT ACCESSION NO.: (9)

3. RECIPIENT'S CATALOG NUMBER: Technical rept.

4. TITLE (and Subtitle): (6) SOFTWARE ACQUISITION MANAGEMENT GUIDEBOOK: VERIFICATION,

5. TYPE OF REPORT & PERIOD COVERED

(14) 6. PERFORMING ORG. REPORT NUMBER: SDC-TM-5772/002/02

7. AUTHOR(s): (10) Harvey/Bratman, Marcia C./Finfer

8. CONTRACT OR GRANT NUMBER(s): (15) F19628-76-C-0236

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
System Development Corporation
2500 Colorado Avenue
Santa Monica, CA 90406

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS:
Deputy for Command and Management Systems
Electronic Systems Division
Hanscom AFB, MA 01731

(11) 12. REPORT DATE: August 1977

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

(12) 74 p.

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)
Computer Program Verification
CPCI Verification
Design Verification
Requirements Verification
Software Acquisition Management
Software Verification
Verification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This report is one of a series of Software Acquisition Management Guidebooks which provide information and guidance for ESD Program Office personnel who are charged with planning and managing the acquisition of command, control, and communications system software procured under Air Force 800 series regulations and related software acquisition management concepts. It provides a review of the software verification practices and procedures employed by industry and set forth in relevant DoD and Air Force regulations,

next page 4/8

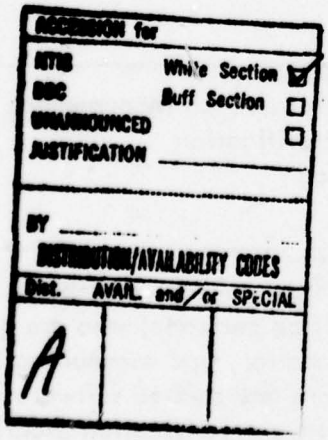
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

339 900

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (con't)

specifications, and standards. It specifically: defines "verification;" describes the software related planning, system engineering, and testing activities, carried out by the Program Office and the contractor, which lead to Computer Program Configuration Item (CPCI) verification; and references specific software techniques and tools required to CPCI verification.



PREFACE

The Verification Guidebook is one of a series of Software Acquisition Management (SAM) Guidebooks intended to help ESD Program Office personnel in the acquisition of embedded software for command, control and communications systems. The contents of the guidebooks will be revised periodically to reflect changes in software acquisition policies and practices as well as feedback from guidebook users.

This report was prepared by System Development Corporation (SDC) under the direction of the Computer Systems Engineering Directorate (MCI) of the Electronic Systems Division (ESD), Air Force Systems Command (AFSC). Contributions were made by: Mr. J. Mott-Smith and Captain W. White (ESD/MCI); Mr. J. Trachtenberg (AFALD/AQE); Mr. M. Landes (RADC/ISI); Mr. M. Mleziva (ESD/EN); Mr. M. Zymaris (ESD/DRT); Mr. D. Peterson (The MITRE Corporation); Captain J. Haughney (AFCS/LO); and Mr. G. Gehlauf (AFLC/LOAK).

The Software Acquisition Management Guidebook series is currently planned to cover the following topics (National Technical Information Service accession numbers for those already published are shown in parentheses):

Regulations, Specifications and Standards (AD-A016401)

Contracting for Software Acquisition (AD-A020444)

Monitoring and Reporting Software Development Status (AD-A016488)

Statement of Work Preparation (AD-A035924)

Reviews and Audits

Computer Program Configuration Management

Computer Program Development Specification (Requirements Specification)

Software Documentation Requirements (AD-A027051)

Verification

Validation and Certification

Overview of the SAM Guidebooks

Software Maintenance

Software Quality Assurance

Software Cost Estimation and Measurement

Software Development and Maintenance Facilities
(AD-A038234)

Life Cycle Events (AD-A037115)

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

LIST OF FIGURES

SECTION 1 - INTRODUCTION

1.1 PURPOSE

This Verification Guidebook is designed to assist the Program Office and its Software Director in planning and managing the implementation of software verification concepts and requirements as they relate to military Command, Control and Communications system software acquisition management. It provides a review of the verification practices and procedures employed by industry and set forth in relevant Department of Defense and Air Force regulations, specifications, and standards. This guidebook describes those Computer Program Configuration Item-oriented system engineering and test activities which lead to verification. It:

- Defines the term "verification" and distinguishes it from the terms "validation" and "certification".
- Describes the software-related planning, system engineering, and testing activities, carried out by the Program Office (PO) and the contractor, which lead to Computer Program Configuration Item (CPCI) verification.
- References specific software techniques and tools required in CPCI verification.
- References appropriate Department of Defense (DoD) and Air Force Regulations, Specifications, and Standards (RSSs) that establish the basis for CPCI verification.

1.2 VERIFICATION, VALIDATION, AND CERTIFICATION DEFINED

Verification is CPCI oriented. It begins with system and software engineering activities, which lead to CPCI definitions and to the CPCI Development Specification, and ends with the qualification of the CPCI.

Validation is system oriented. It begins with the System Specification and concludes at the end of System Development Test and Evaluation (DT&E).

Certification is a user-oriented, system-level activity and occurs during Operational Test and Evaluation (OT&E).

Figure 1 illustrates verification, validation, and certification within the context of this guidebook series by showing: (1) the five phases of system acquisition plotted along an arbitrary time line; (2) the major software related products; and (3) arrows relating the products to the baselines against which they are evaluated or tested. Each arrow is labeled to indicate the

specific review test or audit during which the product is evaluated. In addition, the arrows are labeled to indicate which of the three processes is involved (verification, validation, or certification). The following paragraphs define the terms verification, validation, and certification within this context. These definitions also serve to distinguish the subject matter of this guidebook from that of the Validation and Certification guidebook.

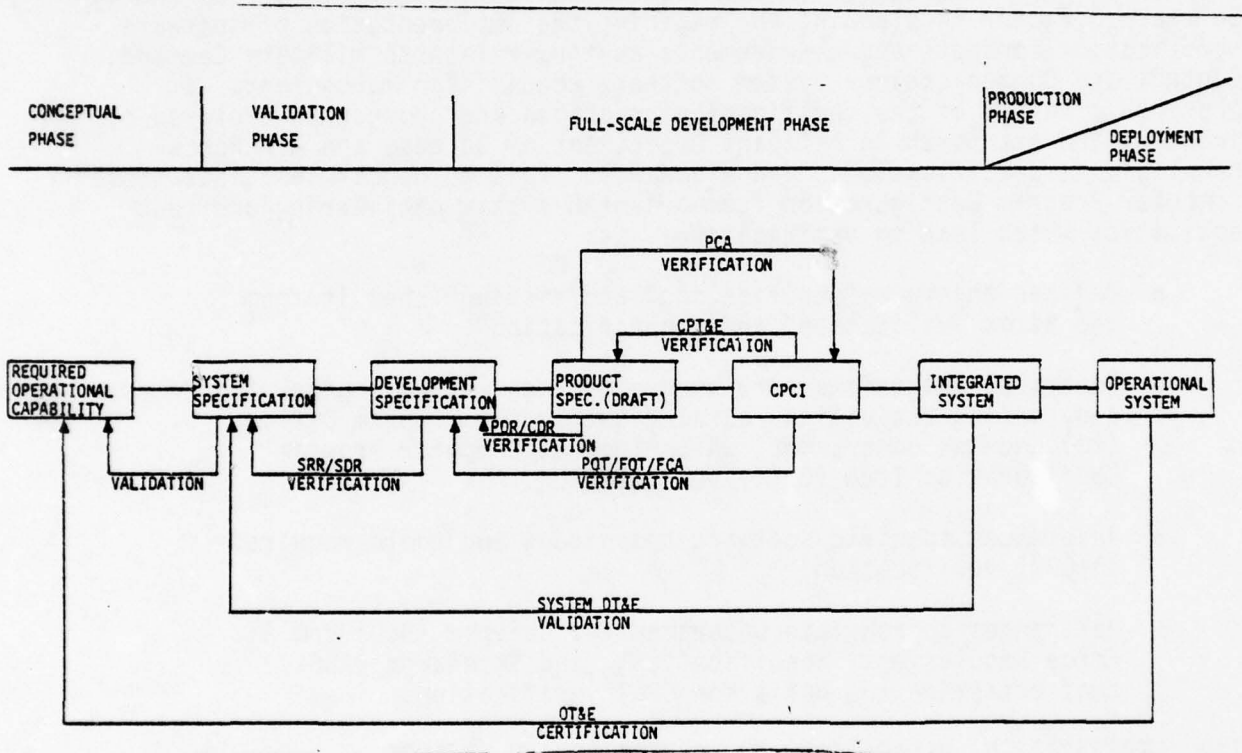


Figure 1. The Scope of Verification, Validation, and Certification

1.2.1 Verification

Verification, as used in this guidebook series, is the iterative process of determining whether the product of selected steps of the CPCI-development process fulfills the requirements levied by the previous step. Specific task areas that make up the CPCI verification process include:

- System engineering analytical activities carried out to ensure that the CPCI Development (Part I) Specifications reflect the requirements allocated from the System Specification (verifying the Development Specification). (See Section 2.)

- Design evaluation activities carried out to ensure that the CPCI design continues to meet the requirements of the Development Specification as the design proceeds to greater levels of detail [Preliminary and Critical Design Reviews (PDR and CDR)]. (See Section 3.)

- Informal testing of the CPCI and its components [Computer Program Test and Evaluation (CPT&E)] carried out by the contractor at his discretion to assist in development, provide visibility of progress, and prepare for formal testing. (See Section 4.)

- Formal testing of the CPCI carried out by the contractor in accordance with Air Force-approved test plans and procedures to verify that the CPCI fulfills the requirements of the Development Specification and to provide the basis for CPCI acceptance by the Air Force [Preliminary Qualification Test (PQT) and Formal Qualification Test (FQT)]. (See Section 4.)

The CPCI contractor is responsible for most of the CPCI verification tasks although the PO monitors and controls his performance by authenticating the Development Specification, participating in design reviews, approving the test documentation, witnessing the execution of formal tests, and approving and accepting test results. The CPCI Development Specification provides the baseline against which the CPCI is verified (Qualified). Verification has the basic Quality Assurance (QA) objective of ensuring that the developing CPCI retains its equivalency to the current baselined specification as design and development proceed to increasingly lower levels of detail. Thus at the System Design Review (SDR), the contractor must show that the requirements to be included in the Development Specification are traceable to the System Specification. At PDR and CDR the contractor must demonstrate the equivalency of each successively detailed design to the baselined Development Specification. During qualification [PQT, FQT, and Functional Configuration Audit (FCA)], the contractor must demonstrate that the coded programs meet the Development Specification requirements. Finally, an audit, the Physical Configuration Audit (PCA), is held to verify that the Product (Part II) Specification is a true representation of the coded and tested CPCI. In summary, verification comprises system engineering and computer programming-oriented evaluation and testing activities carried out at the Computer Program Component (CPC) and CPCI levels by the CPCI contractor and monitored by the PO. (See the "Reviews and Audits Guidebook" for descriptions of the activities to be conducted at FCA and PCA.)

This guidebook discusses verification in terms of activities which are to be performed by the PO. Much of the guidebook is written in terms of verification actions which the PO (either alone or assisted by an independent contractor) must perform. To assist the PO in evaluating contractor proposals, plans and progress, the guidebook also discusses verification activities performed by the development contractor. Figure 2 identifies the major verification-related, products and milestones which are discussed in this guidebook.

VALIDATION PHASE		FULL-SCALE DEVELOPMENT (FSD) PHASE	
Product	See	Product	See
System Engineering & Software Engineering Studies (Technical Reports)	2.1 & 2.2	Development (Part I) Specification	2.3.2
		CPCI DT&E Plans	2.3.3
Development (Part I Specification)	2.2	Computer Program Development Plan (CPDP)	2.3.1
		CPCI Design	3.1 & 3.2
		CPCI DT&E Procedures	3.2.3
		CPCI	4

Figure 2. Major Verification-Related Products

1.2.2 Validation

Validation, as used in this guidebook series, comprises those evaluation, integration, and test activities carried out at the system level to ensure that the system being developed satisfies the requirements of the System Specification. While the validation process has significant software implications, a software validation process, distinct from the system validation process, cannot be isolated since all evaluation and test activities that make up validation are focused at the system level.

Specific validation tasks (see "Validation and Certification Guidebook" for a detailed description of these tasks) include:

- System engineering activities carried out to ensure that the requirements in the System Specification accurately respond to the operational needs called for in the Required Operational Capability (ROC) (validating the System Specification).

- Configuration Item (CI) integration activities (including CPCI integration) carried out to assemble and check out qualified CIs as a fully functioning system (installation and checkout).

- Test Planning and execution activities carried out during System DT&E to demonstrate that the completed system meets the requirements called for in the System Specification (validating the system).

Major software-oriented subtasks can be readily identified within each of the above tasks. Nevertheless, it is not productive to try to define a separate software validation process. To do so implies that the CPCIs qualified during the verification process receive separate and distinct treatment during system DT&E and that some special recourse is available to the PO if the qualified CPCIs do not meet system requirements. Such is usually not the case. However, the PO should certainly plan and carry out system validation in a manner that ensures the comprehensive test and evaluation of the software subsystem.* Furthermore, analysis of system test results may require detailed examination of software performance.

The PO is directly responsible for carrying out the validation program although it is usually a contractor-supported activity.** During the Conceptual Phase the ROC provides the primary baseline for validating the System Specification. The tasks of validating the System Specification, integration, and checkout fall within the system engineering responsibilities of the PO. Validating the system itself is the responsibility of the Test Director. In summary, validation comprises using-command oriented, functionally scoped, system engineering, integration, and testing carried out at the system level by the PO staff, supported as necessary by contractor personnel.

*The software subsystem is the aggregate of CPCIs in the system.
**See AFR 800-14, Volume II, paragraph 5-3c.

1.2.3 Certification

Certification, as used in this guidebook series, refers to the using command's agreement, at the conclusion of OT&E, that the acquired system satisfies its intended operational mission. During OT&E the system has undergone test and evaluation aimed at assuring operational effectiveness and suitability under operational conditions.

1.3 RELATIONSHIP TO OTHER GUIDEBOOKS

This guidebook does not stand alone in providing information on verification. The Overview guidebook establishes a frame of reference for the whole guidebook series. The Validation and Certification guidebook provides more detail on System Requirements Reviews (SRRs) and SDRs. The Reviews and Audits guidebook provides more information on the engineering design reviews and configuration management audits. The Software Documentation Requirements guidebook covers test planning and reporting documentation. Finally, the Configuration Management guidebook provides information on configuration management procedures related to verification, particularly on configuration control during DT&E. An effective verification program must incorporate the concepts presented in all of these guidebooks.

1.4 CONTENTS

The subsequent contents of this guidebook include three sections and three appendixes, as follows:

- Section 2 - Requirements Verification. Addresses requirements verification from initial CPCI definition until authentication of the Development (Part I) Specification and verification of the contractor's CPCI DT&E plan. Discusses contractor activities (2.1); PO verification activities during the Validation Phase, including determination of Validation Phase support products, SRR, and SDR (2.2 through 2.2.3); PO verification activities during the Full-Scale Development Phase, including evaluation of the contractor's CPDP, authentication of the Development Specification, and review of the contractor's CPCI DT&E plan (2.3 through 2.3.3).

- Section 3 - Design Verification. Covers design verification activities which occur during the Full-Scale Development Phase. Discusses contractor activities (3.1); PO activities, including PDR, CDR, and review of the contractor's CPCI DT&E procedures (3.2 through 3.2.3).

- Section 4 - Computer Program Verification. Discusses computer program verification activities in terms of (1) informal testing of the CPCI and its components as carried out by the contractor at his discretion and (2) formal testing of the CPCI as carried out by the contractor in accordance with Air Force-approved test plans

and procedures. Specifically addresses contractor-internal testing, including CPC code and test, CPC incremental-integration testing, and CPCI testing (4.1 through 4.1.3); qualification testing, including PQTs and FQT (4.2 through 4.2.2).

- Appendix A - Support Tools and Techniques for Computer Program Development and Testing. Describes support tools and techniques that aid in computer program development and testing.
- Appendix B - Glossary. Defines terms and acronyms used in this guidebook.
- Appendix C - Bibliography. Provides a list of RSSs, technical books, and papers that are particularly relevant to the subject of software verification.

SECTION 2 - REQUIREMENTS VERIFICATION

This section discusses requirements verification from initial CPCI definition until authentication of the Development (Part I) Specification and verification of the contractor's CPCI DT&E plan. CPCI requirements verification at this time, is focused on the engineering and test planning activities, products, and review points associated with the allocation of performance requirements and the verification of these requirements as stated in the CPCI Development Specification. These engineering activities are normally conducted during the Validation Phase. The engineering process is essentially a decomposition of the system requirements from a higher (user-oriented) level to lower and lower levels of functional, design, and test detail. Requirements Verification is concerned with assuring (verifying) that each succeeding level of requirements is consistent with the previous level of presentation.

Verification is illustrated in Figure 3 and contrasted with validation and certification. The successive development of specifications from the System Specification to the CPCI Development (Part I) Specification is shown with arrows indicating the verification of each product against the previous product.

2.1 CONTRACTOR ACTIVITIES

The main development activities leading from the System Specification are normally the responsibility of the contractor. However, the preparation of the Development (Part I) Specification may also be accomplished by in-house Government resources. The following discussion assumes that the activities associated with Validation Phase engineering activities are performed by a contractor selected by a competitive procurement.

The contractor's major product during the Validation Phase is the Development (Part I) Specification. This specification contains the performance requirements for the CPCI and becomes the authenticated baseline for Full-Scale Development.

The Validation Phase contract should require the contractor to assess the merits of alternative approaches to meeting contractual requirements using trade studies, data collections, analytical modeling, or simulation studies. Standard simulation methods or system-specific methods, such as benchmark, synthetic, and kernel programs, and instruction mixes, may be used to model the critical characteristics of a proposed system on a specific computer, or to evaluate its probable performance on a variety of computers. These techniques are further described in Appendix A, 1.1. The type of study that should be conducted depends upon the type and quantity of data needed,

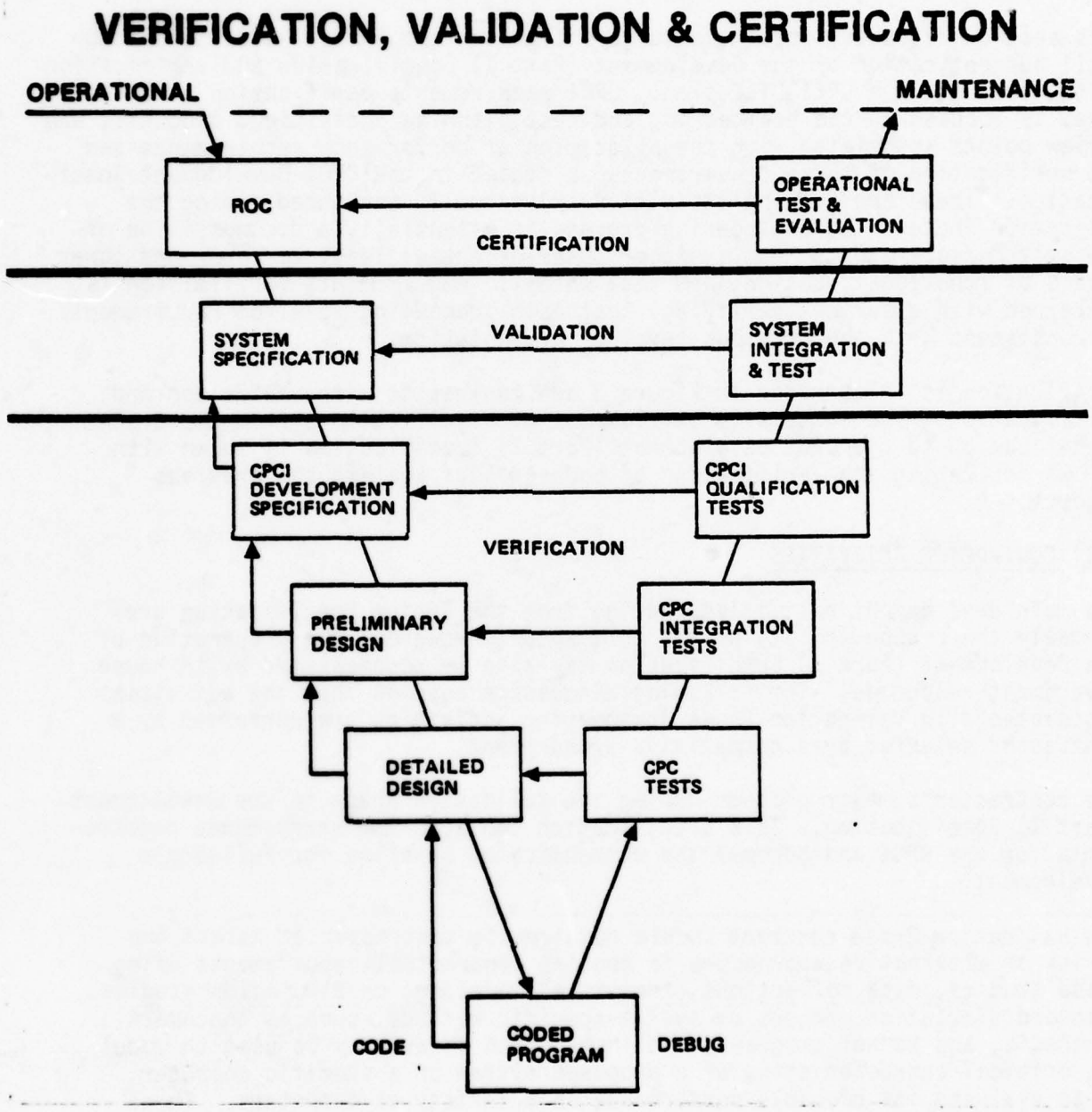


Figure 3. Verification, Validation, and Certification

the complexity of the manipulation/computation being examined, and the time and resources allocated for the study. Engineering studies should normally be conducted only for those alternatives which appear to offer significant payoffs in relation to system objectives, such as total cost, schedules, and operational performance.

2.2 PO VERIFICATION ACTIVITIES (VALIDATION PHASE)

During the Validation Phase, the PO performs requirements verification by reviewing the contractor's system and CPCI software engineering activities aimed at scoping the CPCIs, tracing functions from the system segments to the CPCIs, and detailing performance requirements for each CPCI. The PO's CPCI verification review milestones are the SRR and the SDR. These reviews are further defined in the Software Quality Assurance guidebook and the Reviews and Audits guidebook. System-oriented review activities (validation) are discussed in detail in the Validation and Certification guidebook.

2.2.1 Determination of Validation Phase Support Products

The Validation Phase should begin with a baseline System Specification and a Test and Evaluation Master Plan (TEMP). *However, prior to the start of the* Validation Phase, the PO should determine the need for studies to support the allocation of requirements to CPCIs. Later these studies will provide the basis for software engineering decisions and for CPCI verification. Such studies include:

- Functional-Allocation Trade Studies. A series of studies intended to evaluate the allocation of system requirements to hardware, software, and personnel. They include design optimization, interface simplification, hardware/software tradeoffs, and human engineering studies.

- Sizing and Timing Analyses. The first of an iterative series of activities designed to develop storage and timing budgets based on software performance and design requirements.

- Risk-Identification Studies. Studies that force the identification, description, ranking, and prioritization of risks in the software development process.

All the above studies can be required by tailoring the "Technical Reports" *DID (DI-S-3591) to the specific needs of the program.*

Additional Validation Phase activities which impact the requirements verification process include:

- Software Test Planning Activities. Early test planning decisions directly impact the CPCI verification process. For instance, the software maintenance requirements reflected in the System Specification impact software support tools and documentation. (See the Software Development and Maintenance Facilities guidebook for additional information.)

- The Contractor's System Engineering Management Plan (SEMP). The overall system development process is described in this document. The detailed CPCI plans and schedules should be consistent with the SEMP.

- The Contractor's Computer Program Development Plan. An initial CPDP may be required at the start of the Validation Phase, but the details for a complete CPDP are not available until the end of the Validation Phase. Many verification activities are dependent on the detailed schedules presented in the CPDP. The CPDP should be monitored and updated to reflect any changes in the contractor's software development methodology and plans.

2.2.2 System Requirements Review

The SRR is the first formal review of the Validation Phase. It should be conducted early in the phase, but it may be scheduled as a periodic review meeting. The SRR provides the PO with (1) an early evaluation of the contractor's initial Validation Phase activities and (2) insight into the adequacy of the initial allocation of the data processing requirements to CPCIs. (See AFR 800-14, Volume II, Section 4-9a.) A System Engineering/Technical Direction (SE/TD) contractor may be used at this point in the software acquisition cycle to assist the PO with the evaluation. (See the Reviews and Audits guidebook for a detailed discussion of SRR activities.) Requirements verification activities at this time should ensure that:

- The planned system engineering activities are responsive to the SOW.

- Mission requirements are fully understood by the contractor.

- Total life cycle cost requirements are understood and will be considered by the contractor in defining CPCIs.

- Risk identification and analysis will be emphasized with reference to technical, cost, and schedule factors.

2.2.3 System Design Review

The SDR is the final progress review before the Development (Part I) Specification is completed. At the SDR, the contractor submits technical reports based on the system-engineering tasks completed during the Validation Phase. The primary requirements verification objective at this time is to ensure that the major development risks have been identified and either reduced or eliminated. Engineering analyses or simulations should be verified to demonstrate that a feasible design is available for all identified risk areas. In addition, sensitivity analyses should be verified to evaluate the magnitude of errors in the engineering analyses or simulations which can be tolerated without endangering the goals of the Full-Scale Development Phase. The PO should verify that all research and development efforts conducted to minimize risk are completed before Full-Scale Development is allowed to begin.

The SD should verify the validity, adequacy, and correctness of the material submitted by the contractor at SDR by assuring that:

- Each tradeoff is clearly identified.
- All assumptions are identified.
- All constraints placed on the Development (Part I) Specification are identified and justified.
- Alternative approaches were considered. He should also determine if there should be further analysis to support the alternatives.
- The alternatives were objectively compared and selected.
- The characteristics of the chosen alternatives are traceable to requirements.
- The performance requirements justified by the engineering studies are well documented and properly communicated. (Although the trade studies contain design information, they are not intended to force design approaches upon the Full-Scale Development Phase contractor. Instead, they support the performance requirements contained in the Development Specification.)

In addition, the PO must verify both the equivalency of the evolving Development Specification to the System Specification and the results of system engineering tasks which led to that Development Specification by assuring that:

- The system engineering analyses performed by the contractor support the performance requirements for the selected CPCIs.

- All the defined requirements of the CPCIs are traceable to the System Specification.

- The CPCI to CPCI and CPCI to CI interface definitions are complete, consistent, and compatible.

- All the technical problems (high-risk items) have been identified and solutions have been proposed. Typical risk areas include:

  - Incompatibilities between CPCI performance requirements and hardware capacities.

  - Processing requirements not performed by previous systems or verified by simulations, prototypes, or analytical methods.

  - Entering Full-Scale Development with unproven hardware or unproven support software.

2.3 PO VERIFICATION ACTIVITIES (FULL-SCALE DEVELOPMENT PHASE)

This discussion is concerned with PO requirements-verification activities during the Full-Scale Development Phase. The major PO verification activities include:

- Evaluation of the CPDP.

- Authentication of the Development (Part I) Specification.

- Review of the contractor's CPCI DT&E Plan.

2.3.1 Evaluation of the Contractor's CPDP

The CPDP is a planning document, not a requirements document. However, as a planning document it includes the contractor's description of many activities which relate directly to the CPCI verification process, such as his development and test methods, controls over the developing CPCI, and his detailed schedules and milestones. The SD's evaluation of the CPDP, at a minimum, should verify that:

- Contractor test responsibilities are assigned to an independent individual or group.

- The contractor's test methodology identifies the tools to be used and states their purpose.

- The development and support tools described in the CPDP are the same as those listed in Section 4 of the Development Specification and in the CPCI DT&E plan.

- The sequence of test activities is identified and scheduled.
- The test sequence is compatible with the sequence of design and code; and with the expected availability of test support software and equipment.
- There is a plan for maintenance of all equipment needed for verification.
- The test schedules are consistent with the estimated size and complexity of the functions to be tested.
- The CPDP documents the contractor's plan for controlling the developing CPCI.
- The CPDP documents how the contractor will control and account for problem reports and their resolution.
- The CPDP identifies how the contractor will provide visibility into his CPT&E activities. (Provision for programmer notebooks with up-to-date status is one effective method for providing visibility.)

- The contractor's proposed design methodology is clearly explained. (For example, just stating that he will perform top down design is not sufficient. He should show how he will monitor his down design and verify that it has been met.)
- The CPDP shows how the contractor will verify CI timing requirements. The CPDP should indicate the contractor's plan if timing requirements are not met.
- The CPDP provides a plan for growth, modularity, and ease of modification.
- The CPDP shows show the contractor will verify that his documentation is correct.

2.3.2 Authentication of the Development (Part I) Specification

Authentication of the Development Specification is the PO's performance requirements verification prior to baselining. Preferably this activity should be accomplished prior to beginning the Full-Scale Development Phase because the Development Specification is the contractual baseline for Full-Scale Development. However, in no case should authentication be allowed to extend past PDR. See the Requirements Specification guidebook for a complete description of the purposes, uses, and contents of the CPCI Development Specification.

In performing requirements verification of the Development Specification, the SD should assure that:

- The specification clearly distinguishes between each requirement and information that does not constitute a requirement. (All "shall" statements should be requirements.)
- The specification demonstrates the contractor's understanding of the primary mission by his statements of CPCI requirements.
- The performance requirements are sufficiently detailed so that each function to be performed by the CPCI is fully described in performance terms. For example:
  - All CPCI inputs and outputs must be completely and fully described so they can be designed and tested.
  - All timing requirements must be explicitly stated, both individually and for the CPCI as a whole.
  - Error processing logic must describe CPCI performance when improper, incorrect, or out of range inputs are received.
- Interface compatibility has been established between the CPCI and other CIs.
- All system limits and capacities are compatible with the System Specification.
- All man-machine interfaces impacting the CPCI are fully described.
- All adaptation data requirements are defined (e.g., radar positions, air base locations, etc.).
- All required new technology or design methodology has been proven during the Validation Phase.

- Test facilities are responsive to the requirements identified in Section 4 of the CPCI Development Specification.

2.3.3 Review of the Contractor's CPCI DT&E Plan

The CPCI DT&E (formerly Category I Test) plan provides the basis for CPCI qualification testing (PQT and FQT). Data Item Description (DID) DI-T-3703 describes the format and the content of the CPCI DT&E plan. The test plan should be delivered at the end of the Validation Phase. It should be updated prior to PDR.

In verifying the CPCI DT&E plan, the SD should assure that:

- All CPCI performance requirements can be demonstrated.
- The qualification test environment is sufficient to demonstrate CPCI performance.
- The plan is written in accordance with the DID, as tailored for this specific contract.
- If any portions of the CPCI have been excluded from the test plan, the reasons for their exclusion have been satisfactorily explained.
- The plan conforms to the requirements of Section 4 of the CPCI Development Specification.
- The qualification requirements of the test plan have been related to specific requirements as stated in Section 3 of the Development Specification.
- The methods for determining performance are identified (e.g., analysis of recorded data, examination of displays, etc.).
- The test plan identifies the requirements to be satisfied during CPT&E, PQT, FQT, or System DT&E. (Requirements which cannot be demonstrated until System DT&E should also be included in the System DT&E plan).
- The location and schedule for each test is identified.
- Any limitations on test implementation and accomplishment of test objectives are described. If any of the limitations impair the validity of the tests, then qualification of the CPCI should be delayed until a sufficient test environment can be obtained to verify its performance.

- Satisfactory plans for the preparation of input data are provided and that all necessary input methods and tools are described and will be available when needed.
- Test responsibilities are clearly defined.
- All necessary personnel, facilities, equipment, and related CPCIs are specified and that their availability is scheduled.
- Procedures are established to:
  - Revise or update the test plan
  - Document and revise the test procedures
  - Document the test reports
- The planned test schedules are compatible with the development schedule.

SECTION 3 - DESIGN VERIFICATION

This section covers design verification activities which occur during the Full-Scale Development Phase. Software design activities actually begin during the Validation Phase as system or software engineering studies which support the feasibility of the Development (Part I) Specification. Verification of the initial design is an integral part of the requirements verification activities which occur at SDR. Although the design developed during the Validation Phase is usually included in the Full-Scale Development Phase design, it need not be. The Full-Scale Development Phase contractor is usually contracted at the Development (Part I) Specification level and is responsible for developing the design to satisfy the requirements of the Development Specification. Design verification is performed by both the contractor and the PO.

3.1 CONTRACTOR ACTIVITIES

The contractor's primary engineering activities during the Full-Scale Development Phase are aimed at producing the CPCI design. The PO reviews the developing design at the overall-CPCI level during PDR and at the detailed-CPC level during CDR. The contractor retains responsibility for his design throughout the reviews and can change the design as necessary to meet his contractual Development (Part I) Specification requirements.

3.2 PO ACTIVITIES

This discussion is concerned with PO design verification activities which include:

- PDR
- CDR
- Review of the contractor's CPCI DT&E procedures

3.2.1 Preliminary Design Review

CPCI design is based upon the performance requirements in the Development (Part I) Specification. Initial CPCI design results in the development of the structure or architecture of the CPCI. It defines the CPCI, and describes the sequence or priorities of CPC operations. This initial design is reviewed by the PO at PDR. Following PDR, the contractor develops the design of each CPC to a level of detail sufficient for program coding to begin. This detailed design is reviewed by the PO at CDR. (See the Reviews and Audits guidebook for a detailed discussion of these engineering design reviews.)

While the CPCI representation differs, the objective at both PDR and CDR is essentially identical, i.e., to determine that:

- The design approach for the CPCI will satisfy the performance requirements of the Development Specification.
- The functional interfaces between the CPCI and other CPCIs are complete and correct.
- The design is compatible with the timing requirements.
- The test requirements and test tools are sufficient to determine if the CPCI meets the requirements of the Development Specification.

The PDR provides the PO with a formal technical review of the contractor's progress in developing a design approach for a CPCI. For verification purposes, the PO is concerned with the identification of CPCs and the allocation of CPCI performance requirements to the CPCs.

The contractor presents his overall CPCI design at PDR. If called for in the contract, the design may be documented in a Subsystem Design Analysis Report* or in a draft of selected sections of the Product (Part II) Specification (Sections 1, 2, 3, 3.1; see Appendix VI of MIL-STD-483), otherwise, the design may be described in informal working papers or by contractor presentations.

The design information** presented by the contractor should include:

- An identification and description of the CPC structure of the CPCI, including the functional description of each CPC and input/output data.
- A requirements traceability matrix.
- Detailed storage allocation charts.
- Data base structure and organization.
- Studies to verify sizing, timing, and computational accuracy of CPCI elements.
- Updated CPCI DT&E plans.

*See Software Documentation Requirements guidebook for a description of the Subsystem Design Analysis Report (DI-S-3581).

**See Software Maintenance guidebook for PDR review of features which facilitate the development of maintainable software.

The PO should have sufficient time to prepare for the PDR (review material should be available at least two to four weeks prior to PDR)*. The PO may identify problem areas to be explored and suggestions for additional studies, but since the PDR is a review of the design under the control of the contractor, the PO's suggestions should not be interpreted as direction or approval of the contractor's design approach, but rather approval of a successful PDR. For options available to the PO at PDR, with regard to approval/contingent approval/disapproval, refer to MIL-STD-1521A, Section 4, Paragraph 4.2.4, and Appendix C.

The results of the PDR should include identification of deficiencies in the CPCI design approach. All results and conclusions must be documented in the minutes of the PDR. PO design verification at PDR should be based on determining if:

- All the requirements of the Development (Part I) Specification have been addressed. (The allocation of each Part I performance requirement to one or more CPCs should be shown.)
- The requirements to be satisfied by each CPC have been identified.
- The CPCI design is of sufficient detail for detailed CPC design to take place.
- The CPCI design and the development support tools are compatible.
- The structure of the CPCI data base is established, including organization and intended functional uses.
- The CPCI control flow is established:
  - Processing priorities are described
  - Startup/startover design is shown
- The design methodology described in the CPDP is reflected in the contractor's design.

*This time may vary with the complexity of the CPCI.

3.2.2 Critical Design Review

The CDR is a formal technical review, or series of reviews, held upon the completion of the detailed design of each CPCI, or an increment of related CPCs in the case of a large and complex CPCI. The purpose of holding incremental CDRs is to minimize the amount of time required to develop a large, complex CPCI by reviewing the detailed design of an increment of related CPCs when ready, rather than waiting until the design of the entire CPCI is ready for review. Incremental development of functional areas also allows design, code, and test activities to be scheduled separately, resulting in better control and visibility in the development of each increment. The primary purpose of the CDR is to establish compatibility of the CPCs with the CPCI design structure presented at the PDR. The successful completion of the CDR allows the development to continue with CPC code and test activities.

The following information should be available to the PO to conduct a satisfactory CDR*:

- Identification and description of all modules, including functional description, input/output data, range of values, data files required, and internal data descriptions.
- A detailed description of interfaces, including module-to-module within a CPC, module-to-module among CPCs, and modules within a CPC to external interfaces.
- Detailed flow charts or their equivalent (e.g., HIPO diagrams) for each CPC.
- A detailed requirements traceability matrix showing how each Development Specification requirement is allocated to one or more CPCs.
- Internal CPC data structure and organization.
- Approved ECPs (since PDR) and the contractor's evaluation of their effect on the design approach and the development schedules.
- Test procedures to accompany the test plan, if not submitted prior to CDR.
- Updated sizing and timing estimates for each CPC or each program module.

*Further technical review questions for CDR, with respect to the development of maintainable software, are contained in the Software Maintenance guidebook.

As for the PDR, the PO and the review team must have sufficient time to prepare for the CDR. Since a successful CDR signals the beginning of the coding and testing process the PO must determine whether the contractor's design is compatible with the design presented at PDR.

Design verification at CDR is often tedious since it is and should be directed at a detailed analysis of the design. The contractor should have preceded CDR with a series of design walkthroughs conducted by each designer and performed by one or more technical reviewers. To the extent feasible, the PO should also conduct very detailed CDRs, because any design deficiencies found and corrected prior to the start of coding will save significant time and effort during later test and integration activities.

Design verification at CDR should reflect the following goals:

- The design presented at CDR should be a refinement (i.e., more detailed) of the design presented at PDR.
- The design should be suitable for the problem posed by the requirements, e.g., the CPC timing and sizing estimates should be indicative of success in meeting the design constraints of the interfacing equipment and the spare capacity/growth requirements.
- The data base should be completely defined.
- All Development Specification requirements should be allocated to specific modules within CPCs.
- All requirements for the design of interfaces between CPCs should be addressed.
- The design should be sufficiently detailed to begin coding.

3.2.3 Review of the Contractor's CPCI DT&E Procedures

The CPCI DT&E procedures should be available for review by the time of the CDR for the corresponding functional area. In reviewing the CPCI DT&E procedures, the SD should assure that:

- There is a test procedure for each qualification test. Generally, each procedure will cover one or more functional areas.
- All the performance functions to be tested have been identified. [Reference should be made to Section 4 of the Development (Part I) Specification.]

- The method for determining if each test condition is met is stated, e.g., visual observation, data reduction and analysis, special timing analysis. This part of the test procedure should also be compatible with Section 4 of the Development Specification.

- The location, schedule, contractor, and individual skill-level responsibilities have been established for the necessary briefing, test, debriefing, and analysis activities.

- The procedures have applicable references to the associated: test plan, CPCI Development Specification (specific paragraphs), manuals, positional handbooks, and documentation for support programs or equipment.

- There are procedures (or reference to procedures) for operating the CPCI to be tested.

- The detailed test description is adequate. The PO or the technical reviewer should run through a step by step review of the procedure, should anticipate contingencies, and should ensure that sufficient information (or necessary references) are available for the tests to proceed.

NOTE

Detailed CPCI DT&E procedures can be extremely valuable for continued verification of the CPCI during deployment. However, to retain their value the PO must ensure that they are continually updated as ECPs are installed.

SECTION 4 - COMPUTER PROGRAM VERIFICATION

This section discusses the following computer program verification activities:

- Informal testing of the CPCI and its components [Computer Program Test and Evaluation (CPT&E) carried out by the contractor, at his discretion, to support his development activities, provide visibility of progress, and prepare for formal testing.

- Formal testing of the CPCI carried out by the contractor in accordance with Air Force-approved test plans and procedures to verify that the CPCI fulfills the requirements of the Development (Part I) Specification and to provide the basis for CPCI acceptance by the Air Force [Preliminary Qualification Test (PQT) and Formal Qualification Test (FQT)].

CPCI verification is based on the following documents:

- Development (Part I) Specification (see the Requirements Specification guidebook and Section 2 of this guidebook)

- CPCI DT&E plan (see 2.3.3).

- CPCI DT&E procedures (see 3.2.3).

In addition, particular attention should be paid to the requirements for CPCI verification during:

- Authentication of the Development Specification (see 2.3.2).

- Preliminary Design Review (see 3.2.1).

- Critical Design Review (see 3.2.2).

The entire process of CPCI verification is the reverse of design where analysts start from a global definition of the system and proceed with successive layers of detail, finally resulting in a detailed CPCI design from which coding activities may be initiated. CPCI verification, on the other hand, usually proceeds from (1) the detailed-CPC level in a simulated environment, to (2) the execution of a small increment of functionally-related CPCs, to (3) the operation of all CPCs, together in a live, or nearly live, environment. Top-down programming calls for a variation of this method whereby key control and input handling programs are developed and tested first.

The top-down philosophy calls for CPC implementation to be planned to avoid simulated inputs, where possible. The structure of the entire CPCI is initially represented by stubs which (1) contain very brief non-functional code or (2) may simulate each CPC's operation by performing abbreviated functions. The stubs are replaced as each coded CPC becomes available.

4.1 CONTRACTOR INTERNAL TESTING

Contractor internal testing (CPT&E) consists of CPC code and test (see 4.1.1), CPC-incremental integration testing (see 4.1.2), and CPCI testing (see 4.1.3). CPT&E is the contractor's CPC/CPCI-design shakedown testing. The incremental coding and testing activities of CPT&E may span nearly the entire Full-Scale Development Phase, overlapping with PQTs and terminating when the contractor has completed his internal CPCI testing and is ready for FQT.

CPT&E activities are the contractor's responsibility and there are usually no contractual constraints on the methodology employed by the contractor. However, there are several ways for the SD to gain visibility into contractor progress during CPT&E. The first and most common way is through effective use of PQTs which can be scheduled throughout CPT&E (see 4.2.1). Contractor-delivered PQT and FQT plans, procedures, and reports provide further visibility. Other ways include an on-site PO representative with specific access to specific personnel, visibility into programmer notebooks, monthly progress reports, and monthly status meetings to report and discuss technical as well as administrative progress.

The process of translating the software design into executable programs is a multi-step operation using many implementation test tools and techniques. The emphasis on the contractor's work during CPT&E is not immediately directed at verification of performance criteria, but instead at implementation of the software design that has been shown by previous design activities to meet the specified performance standards. Since performance criteria are a result of analyses of operational requirements and the proposed design has been correlated with the performance criteria, CPT&E verification activities are directed primarily at determining that the programmed instructions are accurate, consistent, and compatible with the detailed computer program draft Product (Part II) Specification.

CPT&E as presented in this discussion, should be used by the PO to evaluate verification information in the CPDP (see AFR 800-14, Volume II) and as supported by the contractor's QA and configuration management plans. It is applicable to the development of most CPCIs. Although this discussion is aimed directly at the development of application software, the verification activities are the same for support software (compilers, test tools, operating systems, etc.).

4.1.1 CPC Code and Test

CPC coding is the translation of the technical solution of a particular problem into a set of machine-readable instructions for the performance of specific computer operations. The coding process also includes:

- Observance of established, project-specific, administrative coding conventions and standards, such as tagging conventions or comment requirements.
- Generation of global data definitions used by the CPC, where appropriate.
- Adherence to technical, project-supported, programming techniques, such as structured programming or decision table methodology.*

Prior to CPT&E the contractor should have:

- Selected the programming language and the associated language aids (see Appendix A, 3.1.1).
- Ensured that project programmers are familiar with the selected language.
- Ensured that project programmers fully understand the design methodology to be used (e.g., top-down programming).
- Structured the data base and provided the necessary data base support tools [e.g., a communications pool (COMPOOL)].

The adequacy of the contractor's verification activities is indicated by:

- The contractor's manual and automated procedures for obtaining visible outputs for both programmers and management at each step of the implementation process (see Appendix A).
- The manual and automated methods used to incorporate changes in software design, thus affecting the code and test processes (see Appendix A).
- The project tools and aids used in support of coding and constructing the CPCs and data base (see Appendix A, 3.1).
- The project coding conventions and standards and the mechanisms for enforcing those standards (see Appendix A, 3.2).

*See Appendix A, 2.1 and 2.2.

CPC testing (also referred to as subprogram testing or parameter testing) is that testing performed by contractor personnel and directed at assuring the internal accuracy and consistency of each CPC before integration with other functionally-related CPCs. CPC testing begins with each module or unit of code and continues until the entire CPC is developed and tested. Specifically, each CPC must be tested as a unit (see Figure 4) to verify that:

- All possible inputs to the CPC are correctly interpreted.
- Arithmetic and logical functions assigned to the CPC are correctly processed.
- Coding conventions and standards are incorporated in the implementation of the CPC.
- Outputs are correct and consistent with the input data.

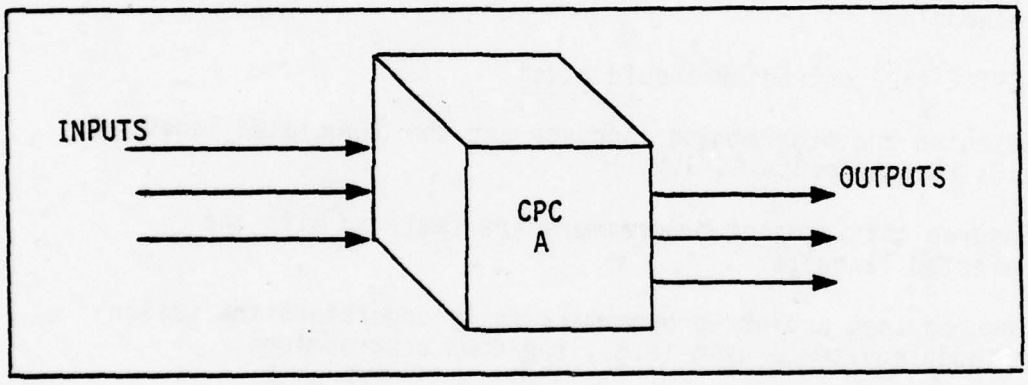


Figure 4. CPC Code and Test

CPC testing activities include:

- Preparing test data
- Compiling or assembling the CPC and reviewing the outputs.
- Running the test data.
- Examining test run results.
- Identifying and correcting errors.
- Repeating each CPC-level testing step until the CPC operates as the programmer's design intended.

CPC testing may also identify requirements for modifications to increase efficiency and maintainability, to meet coding conventions and standards, and to change the program when testing or the contractor's internal audit procedures indicate that program quality is unacceptable.

CPC debugging consists of extracting syntax and logic errors, or "bugs", from the software. During CPC debugging each area of code is tested with sample extreme, and illegal (out of range), data values to ensure that the code operates as it was designed. Early stages of debugging rely heavily upon the programmer's desk-checking of computer-produced listings, despite tools such as traces, dumps, test drivers, test-case generators, and data-reduction programs.* One essential quality of debugging tools at this stage is that they assist the testing process without requiring insertions of large amounts of code into the program which:

- May significantly alter CPC performance.
- May generate additional errors because of the additional code.
- May hide pre-existing bugs until the added code is removed.

The programming methodology used by the contractor impacts the selection of test tools used. For example, top-down development reduces the need for test drivers, whereas bottom-up development generally requires more test drivers.

Debugging on a CPC level is complete when all necessary tests have been executed without error and there is demonstrable evidence that the algorithms are complete and correct. The quantity and quality of the tests used for CPC testing are highly dependent upon the contractor's internal test effort and upon the individual programmer's approach and habits. The test data used for CPC testing are derived from analyses of internal design specifications and simulation of the CPCI's environment. Sometimes a contractor uses an independent programmer or test team for the testing of each CPC. This approach is more often used in later stages of testing, specifically CPCI testing in preparation for PQT and FQT.

Review of programmer notebooks by either the contractor or representative of the PO, if authorized by the contractor, can provide visibility into the status of CPC code and test activities. Sufficient information should be available to:

- Relate detailed CPC development schedules to current status.
- Ensure that design and coding standards are known and followed.
- Relate the design as presented at CDR to the actual CPC design.

*See Appendix A, 3.3.1.

- Ensure that the CPC reflects all approved and scheduled ECPs.
- Verify that the design and programming methods described in the CPDP are indeed being followed.

4.1.2 CPC Incremental-Integration Testing

After successful completion of CPC testing, the CPCs are combined for CPC incremental-integration testing (see Figure 5). CPC incremental-integration testing is directed at resolving design, logic, data definition, and interface errors existing in the combined operation of two or more CPCs. CPC incremental-integration testing focuses on:

- A sequential integration of functionally-related CPCs.
- Using outputs of one CPC as inputs to the next.
- Verifying that CPCs operate as designed and according to performance requirements.
- Conducting dry runs in preparation for PQTs.

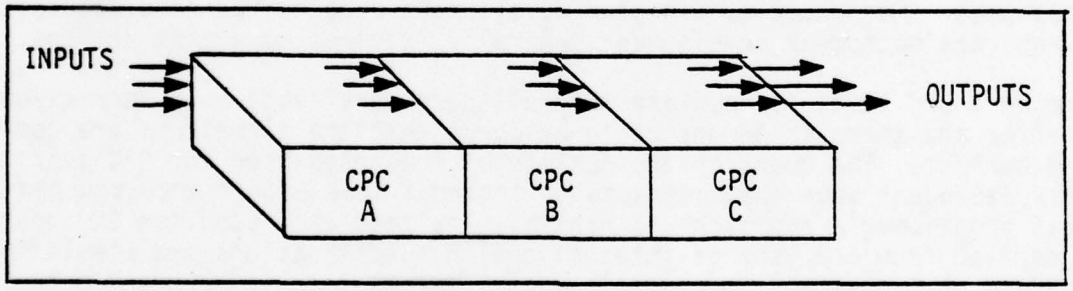


Figure 5. CPC Incremental-Integration Testing

Generally, the CPCs are integrated by combining functionally-related CPCs. In that way meaningful test case data relating to a specific function can be generated by the CPCs themselves, rather than by test drivers. CPC integration can then proceed, incrementing larger numbers of CPCs to provide the input, processing, and output functions needed for complete testing. This approach results in more efficient test-case generation.

In top-down implementation, CPC code and test are accomplished together with CPC incremental-integration testing. Using this approach, the responsible programmer tests each CPC, upon completion of coding, by linking it into the

already developed higher-level CPCI structure and emulating lower-level logic using stubs. In this manner, outputs from a proven CPC are used to provide test inputs for the CPC undergoing test. This method features increased testing of key CPCs in an environment more closely approximating the intended operational environment.

Individual CPC testing should verify that each function was executed correctly with input parameters set first to single values and then to a wide range of values. Similarly, the integrated CPCs should be tested to verify that all functions perform correctly for appropriate single data values, as well as for multiple inputs covering an entire range of applicable data values.

Aids for CPC incremental-integration testing include:

- A Program Production Library (PPL)
- Off-the-shelf routines (operating system and utility library routines)
- Timing and sizing analyses
- Tools to ensure thoroughness of testing
- The contractor's internal-change control procedures

4.1.2.1 Program Production Library

A PPL*, whether automatic or manual, should be used to catalog test cases, program CPCs, load CPCs, and maintain data dictionaries. Using a PPL, maintenance of program versions and test case information by either a programmer or a project librarian can be systematic and relatively simple. Performance data on the evolving CPCI can be readily accumulated and a log of integration and CPCI test runs identifying components of each test can be maintained.

4.1.2.2 Off-the-Shelf Routines

The operating system and the program master utility/library routines associated with the development computer are important aids to efficient CPC incremental-integration testing. Library routines include collections of standard, multipurpose programs used by the operating system to provide efficient hardware/software interfaces of proven mathematical accuracy as needed by the mission software. Should certain system input or output capabilities be unavailable

*See Appendix A, 3.4. See also the Monitoring and Reporting Software Development Status guidebook.

or insufficient for the developing software, auxiliary routines must be provided to augment the existing system. Unless these routines are thoroughly checked out prior to their acquisition, CPC-integration testing may inadvertently become a test vehicle for them.

4.1.2.3 Timing and Sizing Analyses

Although timing and sizing analyses are ordinarily conducted early in the Validation Phase, the CPC integration testing activities are the first opportunity for the contractor to collect functional performance data related to timing and sizing. Sizing data, at this time, are fairly accurate, but timing data are still rudimentary because test inputs are often generated by the computer and timing does not yet reflect the operational environment. However, the contractor should compare these results with his earlier analysis predictions to begin confirming their accuracy and to identify potential problem areas. Since timing and sizing problems often require expensive and time consuming hardware or software redesign solutions, they should be identified as early as possible.

4.1.2.4 Tools to Ensure Thoroughness of Testing

The quality of CPCI-integration testing depends, in part, on the coverage of the conditions tested and the range of values used in the input stream. There are many test tools currently available to improve the quality of CPC incremental-integration testing. However, their suitability to a given test program is limited by the specific computer configuration used and by the specific test requirements. Examples of such tools follow:

- Test-case generators, test beds, and test-data simulators can provide data values that span the range, domain, and distribution of the program's expected inputs.
- Test-output tools include those tools that record, reduce, and analyze the data generated by the program in functionally processing input data.
- Automated flow charts* are sometimes used in the late stages of CPC incremental-integration testing when the development organization must supply detailed flow charts for the Product (Part II) Specification. However, they are more often used in CPCI-qualification testing since changes to program code require a totally new analysis of the syntax by the flow-chart program.

*See Appendix A, 3.4.

- Execution analysis programs*, are generally applied to CPC incremental-integration testing after an increment has successfully been debugged. They are intended to verify the percent of the code tested by accumulating the instruction-execution data of test cases used in the testing process. They identify the code executed by the test(s), the number of test cases run, and sometimes, the range of values applied to specific parameters during each test. Execution analysis programs also assist in determining the need for additional test cases for areas not covered by previous testing, or superfluous code not reached by any other instruction. They also provide management visibility into the amount of code exercised during testing.

4.1.2.5 Contractor Internal-Change Control Procedures

The contractor's internal-change control procedures become especially important during CPC incremental-integration testing. These procedures are documented in the contractor's configuration management plan. (See Configuration Management guidebook.) They are particularly important for verification because:

- All test personnel must know the content and status of the software they are testing.
- The up-to-date status of all problems must be disseminated to prevent wasteful rework by uninformed personnel.
- The status of all problems and PO-approved changes must be known by the contractor's test director so he can determine when retesting is required.

4.1.3 CPCI Testing

CPCI testing by the contractor is primarily a dry run of FQT, but it is a necessary part of CPT&E because it completes the iterative process of testing, correcting, and retesting. If individual CPC and CPC incremental-integration testing has continually emphasized testing to design limits, CPCI testing becomes a device to verify for the contractor's management that the FQT will be passed successfully. Initially, CPCI testing focuses on verifying the total CPCI design. However, the major effort soon shifts toward verifying that the CPCI, including all of its related components, satisfies the requirements of the Development (Part I) Specification.

*See Appendix A, 3.3.2.

The purpose of CPCI testing is to verify that all the components of the CPCI interface together to perform their required functions while not exceeding the limits of tolerances and qualification criteria.

The test plans, procedures, and test data used during the contractor's CPCI testing should closely relate to those submitted for PQTs and the FQT. The test plans, procedures, and related data used in CPT&E are usually not deliverable or available for scrutiny by the SD, unless contractually specified through the inclusion of a Data Accession List on the Contract Data Requirements List (see Software Documentation Requirements guidebook).

The test tools used during the contractor's CPCI tests are the same as those used during CPC incremental-integration testing. However, the use of such tools should be limited since the intent of CPCI testing is to ensure the performance of the CPCI in the operational environment. Obviously, it may still be necessary to use simulated input data generated by tools and processors to record, analyze, and reduce output data. It may also be necessary to use hardware, firmware, or software emulation techniques* for replicating machine or software functions unavailable to the CPCI until a later time. Especially important to CPCI testing, however, is the execution of the CPCI to verify its own performance, not perturbed by unnecessary use of test tools.

The contractor may specify in Section 4 of the Development Specification that a test, or series of tests, run during CPT&E should be run only once because of the high costs incurred in the testing process. This may be particularly applicable to obtaining test results supporting critical timing data. In this situation, the contractor requests that the PO accept CPT&E results as qualification of that requirement in lieu of a PQT or an FQT of that requirement. The contractor has the option of submitting an ECP to request the acceptance of CPT&E results in the event he has not so specified in the Development Specification. The PO may approve or disapprove the request in either case, but should consider such factors as cost, adequacy of test plans and procedures, and criticality of the performance requirement involved when processing the approval/disapproval. [See MIL-STD-483(USAF), Appendix VI.]

The contractor may also defer CPT&E testing of a particular CPC or CPCI interface until operationally-configured equipment is available. This also must be specified in Section 4 of the Development Specification [See MIL-STD-483 (USAF), Appendix VI.]. See 2.3.3 for a list of determinations that must be made by the SD when reviewing CPCI DT&E plans and 3.2.3 for a similar list regarding CPCI DT&E procedures.

*See Appendix A, 3.3.3.

4.2 QUALIFICATION TESTING

Qualification testing is the formal*, contractor CPCI testing which is witnessed by the Air Force. It consists of PQT and FQT.

The test plans, procedures, schedules, and personnel for qualification testing should be planned prior to initiation of CPT&E. Initial test plans for formal testing are submitted with the Development (Part I) Specification and updated after PDR. Prior to formal testing, PO approval of test plans and procedures should consider such factors as:

- Qualification Criteria. Formal testing is conducted to ensure that the CPCI satisfies the qualification criteria stated in the Development Specification and mutually agreed upon by the PO and the contractor. The test plans and procedures used to demonstrate the CPCI should be designed so that each performance feature is observable and measurable. Each test should be designed so that the results can be evaluated by inspection, avoiding ambiguous or subjective evaluation problems between the contractor and the PO. To avoid confusion regarding the achievement of qualification and to avoid redundant testing, qualification tests for any given performance requirement should not overlap.

- Test Documentation. The test documentation supporting formal qualification testing is prepared well in advance of the testing and sets forth the testing scenario, including objectives, inputs, events, expected outputs, etc. The personnel and time needed to produce the support documentation and to perform, observe, and evaluate the tests may be significant. Consequently, the number and amount of scheduled formal tests should be minimized. Careful review and analysis of the draft test plans and procedures may result in substantial changes to the proposed testing activity. Hence, draft test documents should be submitted in sufficient time to allow for careful analyses by the PO and possible revision by the contractor.

- Visibility into Critical Performance Areas. Test plans and procedures must be organized to adequately demonstrate the performance of critical components of the CPCI, e.g., high-risk technological areas and suspected marginal performance areas. The selected order of tests should approximate the order of priorities for development of critical performance and high risk capabilities.

*"Formal" testing is that portion of CPCI testing which is conducted in accordance with Air Force-approved test plans to verify that the CPCI fulfills requirements of the Development Specification. [See AFR 800-14, Volume II, Section 5-3,a,(2)].

- Procedures for Retest. Because a qualification test may fail, disclose a need for redesign, refinement, or reevaluation, or may cause a dispute between the contractor and the PO, a systematic approach for retesting must be established before test plans and procedures are approved and qualification testing is initiated.

4.2.1 Preliminary Qualification Tests

PQTs are planned, scheduled, and performed by the contractor at his development facility to provide visibility into work progress and to demonstrate to the PO that the design meets its performance requirements. PQTs are conducted in accordance with PO-approved test plans and procedures, and test reports submitted in accordance with the Contract Data Requirements List. PQTs are generally scheduled during contractor CPT&E on a sequential basis, often corresponding in sequence to the reviews of the design in a series of CDRs to provide visibility as the CPCs are developed. Each PQT is designed to demonstrate the performance capabilities of a group (or increment) of functionally-related program modules or CPCs. A PQT should demonstrate a CPCI function, especially those functions which are critical to the CPCI. [See AFR 800-14 Volume II; Section 5-3, a,(2),(a)]. A PQT is planned and scheduled by the contractor according to his estimate of when the function can be demonstrated.

PQT procedures must be carefully scrutinized to ensure that PQTs will provide an interim demonstration of the contractor's progress in the development process. PQTs are intended for visibility into time or performance critical CPCs, or functions, not for their qualification. Overly detailed testing, reflecting CPCI design, structure, and internal operation (such as parameter testing at the CPC level), does not provide visibility and may even obscure appraisal of technical adequacy.

The PQT differs from the FQT in two major areas, as follows:

- PQT test coverage may be more detailed and the test results may include intermediate processing data, i.e., data communicated between CPCs, but not a required output of the CPCI. An entire range of data values may be used for a specific parameter to demonstrate functional processing and error processing for illegal values.

- PQTs are conducted at the contractor's development site and may include only minimal hardware/software interface testing. They may also use the contractor's CPT&E test tools and techniques, especially simulation of input data, emulation of hardware or other CPCIs, and output data processors.

4.2.2 Formal Qualification Test

FQT is a comprehensive test of the integrated CPCI, performed by the contractor and witnessed by the PO, to verify that the CPCI meets the performance requirements as stated in the Development (Part I) Specification. FQTs are conducted in accordance with PO-approved test plans and procedures, generally with qualified operationally-configured equipment. FQT normally takes place at a location providing the required equipment capability or at the System DT&E site. It should be completed prior to the beginning of System DT&E. If the required equipment configuration is not available, or if there are performance requirements that cannot be verified in the CPCI DT&E environment, they must be so stated in Section 4 of the Development (Part I) Specification. The requirements are then qualified in the System DT&E environment and the FCA for this CPCI is supplemented by an FQR of the CPCI. For CPCIs that are not dependent upon total system availability, such as support packages, qualification testing is usually conducted at the contractor's site. However, the computer configuration used for qualification at the contractor's site should be sufficiently similar to the operational configuration that no doubts remain about CPCI qualification.

If the CPCI DT&E plans and procedures have been adequately reviewed (see 2.3.3 and 3.2.3) the SD's verification activities at FQT include making the following determinations:

- That FQT proceeds in accordance with the test plans and procedures. If not, each variance should be reviewed to determine its impact upon the qualification tests.
- That all problems are reported and status is maintained. Any deficiencies not corrected prior to completion of FQT should be noted in the FCA minutes (auditing of test results).
- That any features of the test environment which may obscure difficulties in the CPCI are identified. Such features may include:
  - Simulated inputs which are not representative of live inputs.
  - Inadequately verified timing situations.
  - Computer hardware and software different from operational configuration.
  - Insufficient or non-representative site-unique adaptation data.
  - Use, during FQT, of support software that has not been previously qualified.

(Page 44 blank)

APPENDIX A - SUPPORT TOOLS & TECHNIQUES FOR COMPUTER PROGRAM DEVELOPMENT & TESTING

Appendix A is organized to correspond with the stages of verification described in Sections 2, 3, and 4 of this guidebook. It describes the types of support tools and techniques which aid in computer program development and testing*. They must be carefully selected to satisfy the program-specific development or verification requirements and the associated hardware and software configurations. Their primary purpose is to make the software development process easier. However, they also aid the verification process by providing systematic and dependable aids to the system engineering and testing processes.

This appendix discusses the applicability of selected aids to distinct verification and validation tasks. Because of the large number of tools existing for specific applications on specialized computers, this discussion describes generic aids rather than specific tools.

1. REQUIREMENTS VERIFICATION

This discussion addresses evaluation techniques and Development Specification methodologies which can be used to assist performance requirements verification.

1.1 EVALUATION TECHNIQUES

The various evaluation techniques used during the Validation Phase have had varying degrees of success in verifying performance requirements. Such techniques include:

- Simulation
- Performance monitoring
- Synthetic programs
- Benchmarks
- Kernels

Be aware that many of the techniques described differ in the interpretation and use of the results rather than in the evaluation approach used.

*For further information on software tools, see SAMSO TR-75-184 and MITRE WP-21017.

1.1.1 Simulation

Simulation is one of the most powerful techniques currently available for verification of system concepts. Simulation is the process of studying specific system characteristics by the use of models exercised over a period of time and a variety of conditions for the purpose of evaluating alternatives, timing data, system capacities, performance, and constraints within the confines of that system.

For the Conceptual and Validation Phases, it is not necessary to examine all the design details of the system, so a simplified model can be used to gather information directly pertaining to that system's functional performance. The result of the modeling studies should provide sufficient data to verify the specific system concept. However, the simplification benefits of simulation can also be its drawbacks. Some simulations are too simplified to be meaningful. There is currently no universal modeling program capable of simulating any proposed computer configuration for all systems. Also, there is a wide variety of simulation methods, most of which are too costly in time and money to develop and use effectively. Sometimes the structure of required data is complex and a large quantity is required. Obtaining sufficient data to cover the time and conditions to be simulated can be costly and time consuming and may complicate the analysis. However, once the strengths and weaknesses of simulation are recognized it can be a very effective tool to aid in selection evaluation, performance projection, and verification of performance requirements.

1.1.2 Performance Monitoring

Performance monitoring is the process of collecting data on the performance of an existing system for the purpose of evaluating or improving performance or reconfiguring the system. Performance monitoring may also be used in the design of new systems when the instruction set and frequency of use is projected to be similar to an existing system. The process includes both the collection and the analysis of performance data, and can be accomplished by hardware, software, or a combination of both. A hardware monitor is a unit attached directly to a computer's circuitry to obtain and record instruction execution, data transfer, and control information. Hardware monitoring techniques are generally easy to install and use and do not perturb the processes under evaluation. They can also obtain occurrence and duration data of simultaneous events. A software monitor is a computer program that collects performance data on system operation. Software monitoring techniques interrupt the normal programmed procedures to obtain required information at strategic points during the operation of the system under test.

1.1.3 Synthetic Programs

A synthetic program is a set of executable instructions, including I/O operations, files, and operating system resource requirements written for the purpose of representing various computer demands inherent in the system under study. Although synthetic programs offer flexibility in providing a wide range of measurement parameters (e.g., run priorities, projected job mix), the lack of standard synthetic programs makes selection evaluation between proposed hardware configurations difficult. Synthetic programs are also used for performance projection and performance monitoring.

1.1.4 Benchmarks

A benchmark program is an existing operational program used for performance projection or selection evaluation of computer equipment upon which the benchmark is executed. A benchmark, or series of benchmarks, can demonstrate computer operational differences (e.g., CPU performance, I/O channel performance, device management characteristics) while demonstrating software (specifically compiler) speed and execution. Requirements in the selection of the appropriate benchmark must include such factors as the type of selection mix, I/O requirements, and the job mix. Numerous benchmarks may be required to sufficiently evaluate the system hardware and software characteristics.

1.1.5 Kernels

A kernel program is written to evaluate timing information about a specific computer. It represents a partial or complete translation of the time-critical part of an application algorithm (utilizing a complete, or nearly complete, instruction set for a given machine). Kernel programs usually do not include a comprehensive set of I/O operations as they are restricted to user application functions. Kernel programs may be quite large and complex, requiring time for code, checkout, and multiple runs to obtain accurate timing data. They generally provide little information about the effects of the operating system; some compiler data may be available through analysis of object code. The program methodology or code efficiency of the kernel itself must be considered in evaluating the results of kernel operations. The use of kernel programs contributes to hardware evaluation and performance monitoring rather than to software evaluation.

1.2 DEVELOPMENT SPECIFICATION METHODOLOGY

The CPCI Development (Part I) Specification is one result of the system engineering effort of the Validation Phase. There is presently much research work being done in the field of performance requirements specification methodology, including analysis of: problems found to exist in specification documents; a machine-processable language to state system requirements; and procedures to

verify consistency, completeness, and correctness of requirements. There currently exists no reliable tool or technique that can effectively aid in the translation of the system requirements (including the system environment and interactions, performance criteria, and operational functions) into a design-independent document specifying user needs and system data processing functional requirements. Such government owned tools as CARA (Computer Assisted Requirements Analysis) are still under development and evaluation. The problem of adapting a common language, such as English, into a formal language for requirements specification has not yet been solved.

2. DESIGN VERIFICATION

This discussion addresses design aids which can be used in the Full-Scale Development Phase to support the translation of performance requirements [as stated in the Development (Part I) Specification] into a computer program design of sufficient detail to begin CPC coding. Such aids include:

- Design tools and techniques
- Documentation techniques
- Design review techniques

2.1 DESIGN TOOLS AND TECHNIQUES

The design tools and techniques used to support definition of CPC performance requirements, interfaces, and data base definitions, include:

- Simulation
- Top-down design
- Design language
- Decision tables

These tools and techniques are discussed in the following paragraphs.

2.1.1 Simulation

Simulation tools used during CPCI and CPC design are intended to verify that the design will satisfy the performance requirements. Verifying that the CPCI will meet the performance requirements, using analytical methods only, is difficult due to the complex external environment, operating system interactions, and, in some cases, incomplete or inconsistent specifications. The use of simulation or modeling is intended to provide sufficient information during the design process to detect inconsistencies or evaluate alternative equations in the definition and structure of the CPC specifications. Outputs from various simulation techniques are used in trade-off analyses to determine feasible CPCI design.

2.1.2 Top-Down Design

Top-down design is a manual, analytical, design method which is compatible with structured programming techniques. The primary purpose of top-down design is to simplify the overall CPCI structure, thereby reducing the probability of design errors and inconsistencies. Top-down methodology is based on the principle of hierarchical development. That is, a system is composed of successive layers of more and more theoretical operations, beginning with very abstract operations at the user level and ending with primitive, or basic, operations at the machine level. Top-down design implies that the control and interaction of CPCs is defined from the top (the user level) to the bottom (the machine level). Also, the design activity is constrained (and error prone alternatives eliminated) by controlling the interface interactions between levels of CPCs in such a manner so as to "hide" lower level operations and associated data from higher control levels. This has the effect of reducing information transfer interfaces between CPCs. Top-down design is a mechanism used to obtain a clear and consistent functional flow of the CPCI, while attempting to minimize CPC interfaces and indiscriminate use of data. This technique simplifies both the process of design and the verification of design.

2.1.3 Design Language

A program design language is a formal language used to describe the control structure and organization of a program by translating design specifications into computer instructions by a processor. Program design languages are not currently advanced to the state where they are capable of translating a machine-independent data processing problem into a structured set of program modules. However, they may be used for determining control flow design alternatives while documenting the design process as it evolves. The use of a program design language has the potential to assist in the verification of specifications.

2.1.4 Decision Tables

Decision tables are a mechanism which can be used to represent information on program conditions, rules, and actions in a tabular form that can be automatically translated to executable code by a processor. Decision tables are a tabular representation of the design which can be used to clarify the control flow of decision alternatives by presenting the information in a concise and understandable format. However, decision tables are most effective when they represent the conditions of a relatively small application area or algorithm and are not generally used in large, real-time, or multiprocessing software applications.

2.2 DOCUMENTATION TECHNIQUES

A limited number of techniques for design documentation are currently used in the design process, most of which attempt to graphically represent control flow and functional processing. There has been a move away from traditional flowcharting documentation methods with the advent of structured programming techniques, but at the same time there appears to be a clearer recognition of the information required to adequately represent components in the system design.

Design documentation techniques include:

- HIPO (Hierarchy plus Input-Process-Output) Charts. A HIPO chart is a device for representing functional system design in a hierarchical manner. Each graphical representation presents a functional process and its subprocessing relationships, as well as the flow of input and output for each process, or subprocess. HIPO charts depict software functions rather than control flow or data requirements.

- Flow Charts. A flow chart is a graphical representation of a solution to a problem or algorithm in which predefined symbols are used to represent specific functions, sequences of operations, equipment usage, control flow, data manipulations, etc. Flow charts may be used for depicting system, subsystem, or program level design. Design flow charts are distinct from computer-generated flow charts produced from source code. Although structured programming techniques deemphasize the use of flow charts as a design aid, design flow charts still provide an effective mechanism for representing functions and functional relationships within and between computer programs and system components. Flow charts remain one of the most effective methods for representing CPC functional relationships and hierarchies.

- Decision Tables. Decision tables in addition to their design evaluation uses are sometimes used to document program conditions, rules, and actions in a tabular, easy-to-read format.

2.3 DESIGN REVIEW TECHNIQUES

There are two types of design reviews being used by contractors on software development projects, individual and team reviews. Individual design reviews have always been used in software development. Basically, this technique consists of having designers analyze each other's design specification to verify its correctness and consistency. A team review is a more formalized process in which a group of experts rigorously examine the design of a group of functionally-related CPCs to detect errors and inconsistencies.

The use of both individual and team reviews has been an integral part of the software design process for many years, but team reviews are currently receiving renewed attention due to structured programming technology. The purpose of a rigorous design review is to discover potential errors and inconsistencies early in the Full-Scale Development Phase when such errors are relatively easy to correct and when costs associated with error correction are low. The allocation of Development (Part I) Specification requirements on a paragraph-by-paragraph basis to one or more CPCs provides a checklist for use at design reviews. Team reviews may have an additional benefit of providing junior project personnel with a learning mechanism for evolving system design, and management personnel with visibility for verifying work progress.

3. COMPUTER PROGRAM VERIFICATION

The following implementation aids (programming tools, programming standards, testing tools, and project support aids) are widely used in the software industry to support computer program coding and testing activities.

3.1 PROGRAMMING TOOLS

Programming tools are used to translate a program design specification into a set of machine-readable instructions or an organized repository of information, i.e., a data base, used and set by the CPCI. Such tools include:

- Compilers/assemblers
- Data base tools
- Consistency analyzers
- Overlay analyzers

These tools are discussed in more detail in the following paragraphs. Also, related aids that attempt to verify consistency and conformity to the established project programming methodology are discussed.

3.1.1 Compilers/Assemblers

Programming languages and their associated compilers are some of the most significant programming tools available to the contractor for use in the Full-Scale Development Phase. Although the criteria for selecting the programming language do not generally include the quality and quantity of compiler-dependent aids, these aids constitute a large proportion of the tools available to the contractor for the verification process. Some of the more widely used tools and aids associated with programming languages include:

- Set-Use Matrix/Cross Reference Analysis. This tool is a program associated with compilers and provides information on the usage of program labels, tags, data variables, constants, or other program elements. The information usually includes the name, a set-use indicator, and the location(s) in the program where the identified item is set and used. The set-use matrix provides a static trace of data flow. A set-use matrix can be obtained for a CPCI by utilizing sophisticated system monitors which use the compiler-generated output for each CPC as input. This type of set-use matrix, also referred to as a cross-reference analysis, is then generated for all data variables used and set by each of the CPCs in the CPCI. It is also possible to obtain cross-reference information on other system components, such as files and macros.

- Reformatter. A reformatter is a program used to restructure the presentation of source code. Symbolic program modules are input to the program along with reformatting conventions. The reformatter outputs the symbolic program according to the requested reformatting conventions; and the output is the same symbolic program in a more readable format. For example, a reformatter can be used to provide standardized indentation.

3.1.2 Data Base Tools

One of the means for communication between system elements is a globally defined data base which contains information required by CPCs in performing their required functions. Sometimes the structure and contents of the data base are sufficiently complex to require tools to build and maintain the information contained in the data base. Some of the tools associated with a data base include:

- Communications Pool Generator (COMPOOL). A program and associated data definition language that allows commonly accessed data to be centrally defined and controlled. The data definitions are input to the program according to its language specifications, processed (or compiled), and the output is a data dictionary available for use by assemblers, compilers, link editors, and data reduction and execution programs. The data definition language generally requires a parameter description, scaling factors, and sometimes value ranges. The use of this type of tool allows system data definitions to be centrally controlled so that they do not have to be defined by each programmer responsible for CPCs that use system data. Central control of the data base definitions is a feature which enhances the verification process by limiting the scope of possible errors.

- Data Definition Program. A program that provides the capability of controlling central data definitions through which CPCs, written in differing programming languages (PL1, COBOL, JOVIAL, FORTRAN), are able to more easily communicate with each other. It also provides a methodology for making changes to system data so that the new definitions will be consistently reflected in all communicating CPCs.

- Data Base Analyzer. A program which analyzes the usage of data variables by CPCs and indicates whether the CPC inputs, uses, sets, or outputs the variable. This program is similar to a set-use matrix or cross-reference analysis.

3.1.3 Consistency Analyzer

At least the following types of consistency analyzers exist to aid analysis of data used by computer programs:

- Unit Consistency Analyzer. A tool which analyzes the syntax of program modules written in a specific programming language to verify consistent usage of globally defined data elements by that module. The purpose of the tool is to ensure that the set/use of parameters by each CPC is consistent with the system parameter definition.

- Interface Consistency Analyzer. A tool that audits the definitions and declarations of module interfaces for compatibility and consistency.

3.1.4 Overlay Analyzer

This type of tool can be used either during design or in the early steps of coding. Its primary purpose is to analyze core memory requirements to provide information on overlay structure to the programmers. The input to an overlay analyzer is the estimated (or actual) core requirements and a list of external CPC names and data references for each module. The output is a report containing information needed for overlay planning.

3.2 PROGRAMMING STANDARDS

Programming standards should be described by the contractor in his Full-Scale Development Phase proposal and should be finalized as part of the CPCI design activities. Programming standards are used to achieve better quality and more consistent products, while contributing to maintainability, testability, and reliability. The need for programming standards has been obvious for some time, although it has sometimes been difficult to enforce established standards. The following discussion is concerned with specific programming standards and existing tools designed to enforce or audit programming standards.

3.2.1 Specific Programming Standards

A few of the more widely, used programming standards, which can also be automatically audited, include:

- Module Size Limitations. To aid comprehension and reduce complexity, maximum module size standards are established and exceptions granted only when necessary to implement a well-conceived design. The intent of a size limitation standard for modules is to (1) improve readability for both testing and maintenance purposes; (2) simplify the development process; (3) enhance module control; and (4) isolate common code for use by multiple modules. Feasible limits for module size vary, since there is a compromise between ease of reading and ability of implementing logical functions. Structured programming advocates suggest that a module should be contained on one page of output listing since multipage program constructs are harder to follow and page turning breaks concentration. If timing requirements are critical, the size limitations set by the contractor may cause some decrease in program efficiency.

- CPC Organization. To enhance readability and logic clarity, a meaningful unit of source text (a description of a procedure, a macro that performs a clearly defined task, or a data definition) should be kept to one page because indentation suffers between pages. The unit of source program text for each CPC should include and be organized as follows:

  - Initial commentary section describing function
  - Source text for the CPC logic
  - Local (CPC-unique) procedures or in-line routines
  - Local data base definitions
  - Definitions of referenced system data base elements.

- Program Constructs. To enhance readability and eliminate intricate logic that is difficult to verify, only closed logical structures should be employed in the construction of CPCs, if the language permits. Closed logical structures are those which have a single entry and a single exit point. Use of the GOTO instruction is limited to branching within the confines of the constructs.

- Indentation. To increase readability, indentation can be a primary means of imparting structure to the source program listing and, where supported by the program language, can be used to show the flow of control and the scope of definition.

- Naming Conventions. To aid readability, testability, and maintainability, names used in computer programs (procedure or macro names, data identifiers and statement labels) should be meaningful. Naming conventions should uniquely identify each CPCI component and all CPCI symbols (variables, constants, and statement labels) shared by more than one CPC. The naming conventions must be easily understood to achieve maximum benefit.

3.2.2 Existing Tools for Enforcing or Auditing Programming Standards

Existing tools that can be used to enforce or audit programming standards include:

- Code Auditor. A program that analyzes the syntax of a CPC (according to the rules of the specific language) to examine each statement for adherance to established coding conventions. This type of tool is constrained both by the specific language syntax rules and the project-specific programming conventions making it inapplicable for multi-development projects. However, it has been found to be an effective mechanism for evaluating adherance to standards and improving both verification and maintenance activities.

- Structured Programming Precompiler. A program, also called a macro processor, which accepts structured programming constructs not supported by the specific language compiler and translates the constructs into compatible source language statements. The output consists of the altered source code which can then be compiled by the specific language processor.

3.3 TESTING TOOLS

This discussion is concerned with those tools that are applicable to the verification activities inherent in the development of large computer systems. At CPC-level testing, the selection and use of test tools should depend upon the tool providing sufficient information to demonstrate the following:

- The CPC's internal logical construction.

- The CPC's input test case data, including nominal, default, null, critical, maximum, and minimal data values.

- Integrity of the CPC's output data.

- Data base integrity, before and after CPC execution.

- The CPC's instruction-execution frequency and related timing information.

For CPC-incremental integration testing, selection and use of test tools should consider how the tool provides information on the following factors:

- CPC interface integrity.
- Input data that is representative of the actual or live data.
- Instruction execution frequency and timing data.
- Core allocation data.
- Data base integrity, before, during, and after CPC operation.
- Output data integrity, such as message and display formats.

The following discussion is concerned with specific aids and is presented in terms of module and CPC-level, CPC-incremental integration, and CPCI testing aids.

3.3.1 Module and CPC-Level Testing Aids

Module and CPC-level testing aids, or debugging tools, are designed to help the programmer locate an error in program code that causes abnormal behavior or termination to occur with a given set of inputs. Debugging aids assist in tracing the execution of software by allowing the examination of the contents of machine registers and memory representing the operational environment in which the software will be executed. The following aids are frequently used for module and CPC-level testing:

- Trace. A computer program used to record data on program execution and machine environment. Data may be collected when selected portions of code or a selected class of instructions operate.
- Dump. A program which outputs all or selected portions of memory after program operations or at specified points in program operation.

- Driver. A computer program which provides inputs for other programs by simulating its operational environment. Drivers range in complexity from a simple sequence of calls to environment simulation routines or complex data generation and simulated time-dependent operations.

- Data Reduction Programs. A program that translates machine output into a format more easily read by project personnel. In some cases, these types of programs subject machine output to statistical or analytical analyses before outputing the listing.

- Test-Case Generator. A program, or set of manual procedures, designed for the purpose of preparing test data for a specific piece of software. Most test-case generators involve some statistical algorithms for frequency distributions or random number generators for generating a wide distribution of input data values. The specifications for a test-case generator are dependent on the specifications of the software for which data is generated, such as range of values, variable types, error conditions. (This class of tools is separate and distinct *from* automatic test-case generators which are still being researched and are not applicable to C^3 systems.)

3.3.2 CPC-Incremental Integration Aids

Many of the tools discussed for module and CPC-level testing are also used for CPC incremental integration testing, such as dumps, data reduction programs, and test-case generators. Additional tools used for CPC integration testing concentrate on verifying that the basic algorithms operate together correctly by displaying information derived from analysis of the CPC's external specifications as defined in the draft Product (Part II) Specification. The following additional tools are used for CPC integration testing:

- Automatic Execution Analysis. A program that analyzes the syntax of a CPC to instrument the source code. Instrumentation is the process of generating and inserting instructions at strategic program locations. The modified program is then compiled and linked with the recording routines. The instrumentation is transparent to the programmer. The CPC is executed with user-supplied test case data and the execution of the CPC is dynamically recorded via the instrumentation. The output data from this type of tool describes the execution frequency of each statement and sometimes includes information concerning input data processing. The output

data are used to generate a more exhaustive set of test cases, or to identify code that is inefficient or superfluous. It does not prove the program correct in any way, but it does provide an indication of the amount of testing applied to the CPC. The output generated by automatic, execution-analysis tools, for a moderately sized program with a minimum set of test cases, takes time to obtain, analyze, and understand. An instrumented program may take as much as 50-100 percent longer to operate on a single test case than the non-instrumented version of the program on the same test case. However, automatic execution analysis provides a quantitative measure of the percentage of a computer program which was tested.

- Dynamic Analysis of System Structure. A program which outputs listings of the CPCI or subsystem structure when it is prepared for execution, such as link editors and loaders. The output provides information on the contents of each load CPC, by specifying the external references made by each CPC within the CPCI.

3.3.3 CPCI Testing Aids

In addition to the tools discussed in 3.3.2, the following tools can be used to assist in CPCI testing:

- Emulators. Hardware and/or micro-code used to permit one computing system to execute computer programs written for another system. Emulation is a technique which allows the performance of each instruction to replicate the characteristics of the original machine, except for speed of operation. Emulation differs from simulation in that in simulation there is not necessarily an identical set of program instructions and/or common instructions are not necessarily executed in the same precise manner. Emulation may also be used in earlier stages of testing when the target computer is not available. In this case, module-level testing and integration testing may be performed on a host machine emulating the target machine.

- Operating and Performance Measurement Tools. These tools require that the parameters impacting individual CPCI performance, as well as the interaction and dependence of those parameters upon each other, be identified so that the tools are able to measure specific performance characteristics such as operating time, core/peripheral storage transfer requirements, and memory used. Some performance measurement tools also contain algorithms for processing and analysis of the data. (See 1.1.2.)

3.4 PROJECT-SUPPORT AIDS

The development of large complex systems requires another set of tools to support analysis, programmers, and management, and indirectly support the verification process. Project-support aids are a combination of technical and managerial procedures and may be manual, automatic, or a combination of both. Project-support aids include:

- Program Production Library. A program production library (PPL) is a system of administrative procedures and files designed to establish and control computer program and test case files, enforce established programming standards, and provide information and visibility for both project management and programming personnel. The use of the PPL helps to coordinate the status of CPCs under development, while also helping to automate configuration control procedures. This is accomplished by storing program modules and test data in a data base and maintaining status and control records on the contents of the data base.

- Project Monitor. An automatic or semi-automatic tool used to provide management, planning, and control information of software under development. These tools are used to build a project data base containing detailed schedule information for collecting and reporting the status of program development. They also are used to keep track of different components of the CPCI, describe their interrelationships, and monitor the progress of their completion. Schedules for CPCs may be plotted, and error reports and resolutions may be tracked. The relationships between specifications, CPCIs, CPCs, test requirements, and test plans can be maintained in the data base and reported as traceability verification matrixes. Matching of test plans and procedures with requirements may also be accomplished. Project monitors are used to provide the contractor with visibility on work progress for the evolving CPCI, but they are sometimes cumbersome and expensive to use because they require detailed and current input data on all system elements to be effective.

- Chief Programmer Team. A contractor organizational concept used in Full-Scale Development which structures job assignments by individual specialization and clearly defines the relationships between team members. Although primarily directed at software development, the chief programmer concept aids verification through its internal review and testing activities. The team is headed by a highly competent chief programmer, whose principal job is to design, code, and test the critical segments of the code and to allocate specific programming assignments. A

backup programmer may assist him in the design of the program and act as an evaluator, but is not held responsible for the code. A program secretary is responsible for maintaining the project records, project notebook, and the PPL. Other programmers and analysts perform duties as designated by the chief programmer. The entire team usually consists of five to nine people. The software produced under this concept is the shared responsibility of the team as a whole. The chief programmer concept features visibility into work assignment and communication between team members.

- Automatic Flowcharters. A program that analyzes the syntax of a program in a specific language to graphically represent the control flow of the source code. Some automatic flowcharters incorporate programmer comments in the graphic display outputs. Most automated flowcharters do not analyze usage of data variables, although they often reflect where data is set and used.

APPENDIX B - GLOSSARY

This appendix consists of (1) definitions of major terms used throughout this guidebook and (2) acronyms and abbreviations used herein.

DEFINITIONS

Certification. As used in this guidebook, certification refers to the using command's approval, at the conclusion of OT&E, that the acquired system satisfies its intended operational mission.

Computer Program Component (CPC). A functionally or logically distinct part of a computer program distinguished for purposes of convenience in designing and specifying a complex computer program as an assembly of subordinate elements.

Computer Program Configuration Item (CPCI). A computer programming end product whose development and subsequent modification is subject to configuration management.

Computer Program Development Plan (CPDP). The CPDP is a plan that identifies the actions needed to develop and deliver computer program configuration items and necessary support resources.

Computer Programming Test and Evaluation (CPT&E). Tests conducted prior to and in parallel with preliminary or formal qualification tests. These tests are oriented primarily to support the design and development process. (AFSCM/AFLCM 310-1).

Critical Design Review (Computer Program). A formal technical review of the design as depicted by the specification and flow diagrams, sufficiently detailed to enable the programmer to code, compile, and debug a computer program, to assure that design requirements have been met before coding begins.

Development (Part I or Type B5) Specification. A document which specifies the requirements pecular to the design, development, functional performance, test, and qualification of the configuration item. It establishes performance criteria and test criteria for which the program shall be designed/developed [MIL-STD-483(USAF)].

Development Test & Evaluation (DT&E). That testing and evaluation of individual components, subsystems, and, in certain cases, the complete system, which is conducted predominantly by the contractor.

Formal Qualification Review (FQR). The test, inspection, or analytical process by which products at the end item or critical item level are verified to have met specific procuring activity contractual performance requirements (specifications or equivalent). This review does not apply to requirements verified at FCA [MIL-STD-1521A(USAF)].

Formal Qualification Tests (FQT). A formal test conducted in accordance with the Air Force-approved test plans and designed to be a complete and comprehensive test of the CPCI prior to FCA. It is conducted after the design process culminates (AFR 800-14, Vol. II).

Functional Configuration Audit (FCA). A formal audit to validate that the development of a configuration item (CI) has been completed satisfactorily and that the CI has achieved the performance and functional characteristics specified in the functional or allocated configuration identification.

Physical Configuration Audit (PCA). A technical examination of a designated configuration item (CI) to verify that the CI "as built" conforms to the technical documentation which defines the CI.

Preliminary Design Review (PDR). A formal review of the preliminary design of a system functional area or a configuration item to establish system compatibility of the design, identify specific engineering documentation and define physical and functional interface relationships.

Preliminary Qualification Tests (PQT). A formal test conducted in accordance with Air Force-approved test plans and designed to be an incremental process which provides visibility and control of the computer program development during the time period between CDR and FQT. A PQT should be conducted for those functions which are critical to the CPCI (AFR 800-14, Vol. II).

Product Specification. A document or series of documents which contain the detailed technical description of the CPCI as designed and coded. It is a complete description of all routines, limits, timing, flow, and data base characteristics of the computer program, limits, timing, flow, and data coded instructions. Equivalent to "Part II CPCI specification" or "Type C5 specification".

Program Production Library (PPL). A group of manual or automated procedures used to control and keep records of the developing software.

System Design Review (SDR). The SDR is conducted to evaluate the optimization, correlation, completeness, and risks associated with the allocated technical requirements.

System Engineering Management Plan (SEMP). The SEMP is a comprehensive plan on how the contractor will manage and conduct his integrated engineering effort.

System Requirements Review (SRR). The SRR is a system engineering review to ascertain the adequacy of the contractor's efforts in defining system requirements. It will be conducted when a significant portion of the system functional requirements has been established.

System Specification. A document which states all the necessary technical and mission requirements in terms of performance, allocates requirements to functional areas (or configuration items), defines the interfaces between or among the functional areas (or configuration items), and includes the test provisions to assure the achievement of all requirements.

Test & Evaluation Master Plan (TEMP). The TEMP is an overall plan which identifies and integrates the efforts and schedules of all test and checkout activities to be accomplished in the system development program.

Validation. As used in this guidebook, comprises those evaluation, integration, and test activities carried out at the system level to ensure that the system being developed satisfies the requirements of the System Specification. While the validation process has significant software implications, a software validation process, distinct from the system validation process, cannot be isolated since all evaluation and test activities that make up validation are focused at the system level.

Verification. The iterative process of determining whether the product of each step of the Computer Program Configuration Item (CPCI) development process fulfills all of the requirements levied by the previous step.

ACRONYMS AND ABBREVIATIONS

AFR.	Air Force Regulations
AFSC.	Air Force Systems Command
C^3.	Command, Control, and Communications
CDR.	Critical Design Review
CI.	Configuration Item
COMPOOL.	Communications Pool
CPC.	Computer Program Component
CPCI.	Computer Program Configuration Item
CPDP.	Computer Program Development Plan
CPT&E.	Computer Program Test and Evaluation
DID.	Data Item Description
DoD.	Department of Defense
DT&E.	Development Test and Evaluation
ESD.	Electronic Systems Division
FCA.	Functional Configuration Audit
FQR.	Formal Qualification Review
FQT.	Formal Qualification Test
FSD.	Full-Scale Development
GFE.	Government Furnished Equipment
HOL.	Higher Order Language
MIL-STD.	Military Standard
OT&E.	Operational Test and Evaluation

PCA. Physical Configuration Audit

PDR. Preliminary Design Review

PO. Program Office

PPL. Program Production Library

PQT. Preliminary Qualification Test

QA. Quality Assurance

RADC. Rome Air Development Center

ROC. Required Operational Capability

RSSs. Regulations, Specifications, and Standards

SAM. Software Acquisition Management

SD. Software Director

SDR. System Design Review

SEMP. System Engineering Management Plan

SE/TD. System Engineering Technical Direction

TEMP. Test and Evaluation Master Plan

TR. Technical Report

USAF. United States Air Force

APPENDIX C - BIBLIOGRAPHY

MILITARY SPECIFICATIONS AND STANDARDS

MIL-STD-483(USAF); "Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs;" DoD; 12 April 1971.

MIL-STD-490; "Specification Practices;" DoD; 30 October 1968.

MIL-STD-499A; "Engineering Management;" DoD; 1 May 1974.

MIL-STD-1521A(USAF); "Technical Reviews and Audits for Systems, Equipment, and Computer Programs;" DoD; 1 June 1976.

AIR FORCE AND SUBORDINATE COMMAND DOCUMENTS

AFR 65-3; "Configuration Management;" USAF; 1 July 1974.

AFR 80-14; "Test and Evaluation;" USAF; 19 July 1976.

AFR 800-2; "Engineering for Defense Systems;" USAF; 16 March 1972.

AFR 800-14; "Acquisition Management;" Volume I-Management of Computer Resources in Systems; 12 September 1975; Volume II-Acquisition and Support Procedures for Computer Resources in Systems; 26 September 1975; USAF.

AFSCM/AFLCM 310-1; "Contractor Data Management;" Volume 1, Management of Contractor Data and Reports; USAF(AFSC/AFLC); 30 August 1969.

AFSCM/AFLCM 375-7; "Configuration Management for Systems, Equipment, Munitions, and Computer Programs;" USAF (AFSC/AFLC); 31 March 1971.

AFSCP 800-3; "A Guide for Program Management;" USAF; 9 April 1976.

AFSC DH 4-2; "Computer Program Testing;" Chapter 5-Electronic Systems Test and Evaluation; AFSC; 10 April 1971

RADC-TR-74-300; "Structured Programming Series;" Volume XV, Validation and Verification Study; USAF(RADC); May 1975.

DATA ITEM DESCRIPTIONS

DI-E-3029, AGENDA - Design Reviews, Audits and Demonstrations

DI-E-3108, Configuration Management Plan

DI-E-3118, Minutes of Formal Reviews, Inspections and Audits

DI-E-3119A, Computer Program Development Specification

DI-E-3120A, Computer Program Product Specifications

DI-S-3581, Subsystem Design Analysis Report

DI-S-3591A, Technical Reports

DI-S-3606, System/Design Trade Study Reports

DI-S-30567, Computer Program Development Plan (CPDP)

DI-T-3703, Category I Test Plan Procedures (Computer Program)

DI-T-3717, Category I Test Report (Computer Program)

GENERAL REFERENCES

"A Definitional Framework;" Hertzel, W.C.; Program Test Methods; Hetzel, W.C. (Ed); Prentice-Hall, Inc.; Englewood Cliffs, N.J.; 1973.

"An Overview of Bugs;" Schwartz, J.T.; Debugging Techniques in Large Systems; Rustin, R. (Ed.); Prentice-Hall, Inc.; Englewood Cliffs, N.J.; 1971.

"A Perspective on System Performance Evaluation;" Drummond, M.E.; IBM System Journal, No. 4; 1969.

"Automation Aids for Reliable Software;" Reifer, D.J.; SAMSO Report TR-75 [illegible]; Aerospace Corporation; El Segundo, CA.; August 1975.

"Computer Program Verification/Validation/Certification;" Reifer, D.J.;* TOR-0074(4112-5); Aerospace Corporation; El Segundo, CA.; May 1974.

"Computer Selection Methodology;" Timmpeck, E.M.; Computer Surveys, Vol. 5, No. 4; December 1973.

"Configuration Management of Computer Programs by the Air Force: Principles and Documentation," Searle, L.V., Neil, G.; AFIPS Conference Proceedings; Vol. 30; April 1967.

"Data Requirements for Productivity and Reliability Studies;" Finfer, M. C.; TM-5542/003/01, System Development Corporation; Santa Monica, CA.; June 1976.

"Debugging Under Simulation;" Supnik, R.M.; Debugging Techniques in Large Systems; Rustin, R. (Ed.); Prentice-Hall, Inc.; Englewood Cliffs, N.J.; 1971.

"Developing and Testing a Large Programming System, OS/360 Time Sharing Option;" Scherr, A.L.; Program Test Methods; Hetzel, W. (Ed.); Prentice-Hall; Englewood Cliffs, N.J.; 1973.

"Interim Report On the AIDS Inventory Project;" Reifer, D. J.; SAMSO TR-75-184; Prepared by Aerospace Corp.; 16 July 1975.

"On the Feasibility of Software Certification; "Keirstead, R. E.; Stanford Research Institute Project 2385 for National Science Foundation (Grant No. GJ 36903x1).

"Performance Evaluation and Monitoring;" Lucas, H. C.; Computer Surveys, Vol. 3, No. 3; September 1971.

*The discussions in this guidebook on requirements verification, design verification, and computer program verification differ in coverage from Reifer's CODEVER, SPECVER, REQVER, and SYSVER.

"Precompiler Specifications;" Tinanoff, N.; RADC TR 74-300; Structured Programming Series, Vol. II; IBM; Gaithersburg, MD.; May 1975.

"Program Design Study;" Kraly, T.M., et al; RADC TR 74-300; Structured Programming Series, Vol. VIII; IBM; Gaithersburg, MD.; May 1975.

"Program Production Library Programmer's Guide;" Bratman, H., Cudney, P.F., Johnson, B.G.; TM-5175/600/00, System Development Corporation; Santa Monica, CA.; August 1973.

"Quantitative Aspects of Software Validation;" Rubey, R.J.; Proceedings of International Conference on Reliable Software; Pgs. 245-251; IEEE; April 1975.

"Reliable Software Through Composite Design;" Meyers, G.J.; Petrocelli/Charter; New York, N.Y.; 1975.

"Software Reliability;" Meyers, G.J.; John Wiley and Sons, Inc., New York, N.Y.; 1976.

"Software Requirements Analysis;" Kassiakoff, A., Sleight, T.P.; Paper presented at Conference on Software Management in Defense Systems and Other Federal Programs; ACM/IEEE; 1976.

"Structured Programming: Techniques for Developing Reliable Software Systems;" Bratman, H.; SP-3693; System Development Corporation; Santa Monica, CA.; December 1972.

"Summary Notes of a Government/Industry Software Sizing and Costing Workshop;" ESD-TR-76-166; USAF (ESD); Bedford, MA.; October 1974.

"System Management Aspects of Computer Program Test and Activation;" Henderson, R.L., Searle, L.V.; TM-3361/000/01, System Development Corporation; Santa Monica, CA.; August 1967.

"System Simulation;" Gordon, G.; Prentice-Hall; Englewood Cliffs, N.J.; 1969.

"System Management Applied to Large Computer Programs in BUIC III; Review of Experience;" Searle, L.V., Rosove, P.E., Sydow, E.H.; ESD-TR-69-302; Air Weapons Surveillance and Control SPO; USAF (ESD); Bedford, MA.; 1969.

"The Program Development Process;" Aron, J.D.; Phillipines: Adison-Wesley; 1974.

"The Software Engineering Facility;" Irvine, C.A., Brackett, J.; Document 553-37; Softech; Waltham, MA.; October 1974.

"Validation and Verification Study;" Smith, R.L.; Structured Programming Series, Vol. XV; IBM; Gaithersburg, MD.; May 1975.

"Verification and Validation of Defense and Space Systems Software;" #76.6455.11-002; TRW; June 1976.

COMMENT SHEET

Software Verification Guidebook

Reviewer's Name: Reviewer's Organization:

Comments:

Please return to: Hq ESD/MCIT (Stop 36)
Hanscom AFB, MA 01731