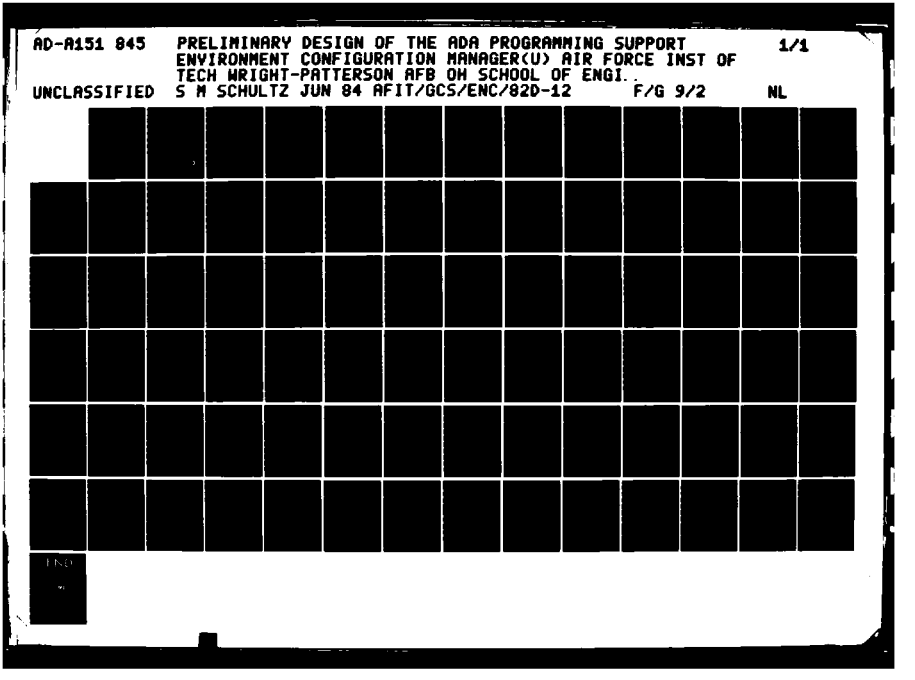
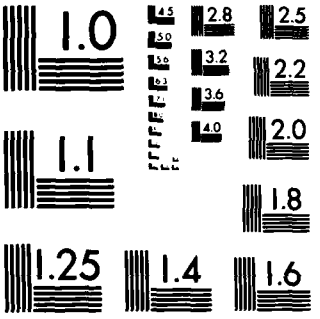

AD-A151 845
PRELIMINARY DESIGN OF THE ADA PROGRAMMING SUPPORT ENVIRONMENT CONFIGURATION MANAGER(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..
1/1
UNCLASSIFIED
S M SCHULTZ JUN 84 AFIT/GCS/ENC/82D-12
F/G 9/2
NL
END



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

PRELIMINARY DESIGN OF THE ADA PROGRAMMING SUPPORT ENVIRONMENT CONFIGURATION MANAGER

THESIS

Susan M. Schultz
Captain, USAF

AFIT/GCS/ENC/82D-12

PRELIMINARY DESIGN OF THE ADA PROGRAMMING SUPPORT ENVIRONMENT CONFIGURATION MANAGER

THESIS

Presented to the Faculty of the School of
Engineering of the Air Force Institute
of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Susan M. Schultz, B.A.
Capt USAF
Graduate Computer Science

June 1984

Preface

The purpose of this thesis is twofold. First, it analyzes how software configuration management is currently practiced, and second, it gives a preliminary design for the Ada Programming Support Environment (APSE) configuration manager.

I choose this topic because SCM is an important, but often overlooked, discipline. SCM offers solutions to such common software project problems as:

1. systems that do less than expected
2. systems that are delivered later than expected
3. systems that are poorly documented

The software configuration manager as a tool of the APSE will make SCM easier to accomplish. Many of its tedious functions will be automated.

I would like to thank my advisor, Major Roie Black, for his help and encouragement throughout this project. I would also like to thank my readers, Lieutenant Colonel Harold Carter and Major Michael Varrierur, for their suggestions and support.

Susan M. Schultz

Abbreviations and Acronyms

ACI	Allocated Configuration Identification
AFLC	Air Force Logistics Command
AFSC	Air Force Systems Command
APSE	Ada Programming Support Environment
ASD	Aeronautical Systems Division
CA	Configuration Auditing
CC	Configuration Control
CCB	Configuration Control Board
CDR	Critical Design Review
CDRL	Computer Design Requirements List
CI	Configuration Identification
CM	Configuration Management
CPC	Computer Program Component
CPCI	Computer Program Configuration Item
CR	Change Request
CRISP	Computer Resource Integrated Support Plan
DBMS	Data Base Management System
DOD	Department of Defense
ECP	Engineering Change Proposal
FB	Functional Baseline
FCA	Functional Configuration Audit

Abbreviations and Acronyms (Continued)

FCI	Functional Configuration Identification
FQR	Formal Qualification Review
KAPSE	Kernal Ada Programming Support Environment
PB	Product Baseline
PCA	Physical Configuration Audit
PCI	Product Configuration Identification
PDR	Preliminary Design Review
PM	Project Manager
PMP	Project Management Plan
QA	Quality Assurance
SA	Status Accounting
SADT	Structured Analysis and Design Technique
SCM	Software Configuration Management
SDR	System Design Review
SIR	Software Incident Report
SOW	Statement of Work
SRR	System Requirements Review
TAC	Tactical Air Command
T&E	Test and Evaluation
V&V	Verification and Validation

Contents

Contents (Continued)

List of Figures

List of Tables

Abstract

Today the development and maintenance of software are becoming prohibitively costly. With the goal of reducing the cost of producing a software system without sacrificing the quality of it, the Department of Defense (DOD) is sponsoring the development of the Ada Programming Support Environment (APSE). This paper explains the APSE. It also explains the requirements and gives a preliminary design of one of the major tools of the APSE, the configuration manager. The preliminary design of this tool is presented using Structured Analysis and Design (SADT) diagrams. The preliminary design includes only a functional description of the configuration manager. How to implement it is left for further research.

Prior to presenting the preliminary design of the configuration manager, a description of how SCM is currently practiced is given. SCM is divided into four functions. They are configuration identification (CI), configuration control (CC), status accounting (SA), and configuration auditing (CA). SA is the only function that can be completely automated. Therefore, the preliminary design emphasizes the SA function.

PRELIMINARY DESIGN OF THE ADA PROGRAMMING SUPPORT ENVIRONMENT CONFIGURATION MANAGER

I Introduction

Background

Today in projects that involve both hardware and software, a "software problem" exists. Software systems are prohibitively costly. The Department of Defense (DOD) estimates that, in projects that involve both hardware and software, only 15 percent of the total cost is attributed to the hardware. Of the software cost, 70 to 90 percent of the money is used for maintenance and long-term life-cycle support (Ref 10:4). With the goal of producing quality software at a lower cost, the DOD is supporting the development of the Ada Programming Support Environment (APSE). Briefly, the APSE is a project to develop an environment to create software primarily for embedded computer systems. The APSE consists of the new computer language, Ada, and a complete set of automated tools to develop, run, and manage Ada programs. The requirements of the APSE are presented in a DOD document called "Stoneman" (Ref 10) which will be referenced frequently throughout this paper.

One of the APSE tools is the configuration manager. This tool is to automate as many of the Configuration Management (CM) tasks as possible. CM is defined in AFR-65-3 as:

> . . . a discipline applying technical and administrative direction and surveillance to:
> 1. identify and document the functional and physical characteristics of a configuration item

2. control change to these characteristics, and
3. record and report change processing and implementation status (Ref 19).

In the past, CM was applied almost exclusively to hardware. CM was not applied to software primarily because programming was considered to be almost an art, not a scientific method used to build a product. Few design techniques were used and changes were made haphazardly. Today attitudes towards software have matured. A piece of software is considered a product, and many tools and techniques, including CM, are used to produce it.

CM applied to software is called Software Configuration Management (SCM). Although SCM has taken its terminology and structure from CM, differences exist between the two. The differences are caused by the differences in managing software and hardware. These differences are:

1. Software is easily changed in any stage of the life-cycle while hardware is not.

2. Hardware components wear out while a piece of software will not.

3. A production phase is not needed for software. It is easily reproduced.

In order to accomplish the goals of SCM, it is divided into four functions. They are Configuration Identification (CI), Configuration Control (CC), Status Accounting (SA), and Configuration Auditing (CA). CI involves identifying and labeling the software items of the system. This function is important in order to have a reference point

for changes. CC involves methods of controlling changes made to the system. This is accomplished through documentation, procedures, and the Configuration Control Board (CCB). SA involves recording, storing, and reporting the history of the system. Both automated and manual techniques are used. CA involves verifying that the software parts of the system are what they are claimed to be through the use of software audits.

Problem Statement

The prime purpose of SCM is to control the software and its associated documentation throughout the project. In the APSE, all the information relating to a project is stored in a data base. Accomplishing SCM in the APSE will require manipulating data that resides in the data base. The goal of this paper is to give the preliminary design of the APSE software configuration manager.

The SCM functions of CI, CC, and CA can not be fully automated. The CI task requires that a decision be made on how the software system is divided up. How this division is made is influenced by many factors and is unique for each project. CC consists of a set of policies and procedures that project members must follow. These policies are made by management and will differ from project to project. CA is rechecking the work already done. Automated tools will only assist in this task. The success of CA still depends heavily on decisions made by the auditors. In contrast, SA can be fully automated. The task primarily involves storing and retrieving selected data from the data base. Due to the fact that of the four

SCM functions only SA can be fully automated, the preliminary design will stress SA.

Organization

To produce this preliminary design, it is necessary to understand how a system is managed. Chapter II discusses this. In addition, it is necessary to understand exactly what SCM is and how it is practiced today. This information is given in Chapter III. Since the configuration manager is part of the APSE, all the requirements set forth in "Stoneman" must be met. An overview of the APSE and its requirements are given in Chapter IV. The specific SCM and SA requirements are also given. Chapter V presents the requirements in the form of Structured Analysis and Design Technique (SADT) diagrams with a focus on the SA function. The last chapter gives conclusions and recommendations.

II Managing the System

Introduction

This chapter gives an overview of managing a system. The phases of the system life-cycle are defined and the major events that happen in each are explained.

Processes

In the study of how systems are built, it has been recognized that three different processes exist. Bersoff, Henderson, and Siegel (Ref 8) label them as:

1. Planning, direction, control.
2. Execution.
3. Evaluation.

Planning, direction, and control is done by the managers. As the name suggests, it involves developing a plan of attack, telling who to do what, and continually checking that the plan is proceeding as expected. The execution process is done by the "doers," the people who actually design and build a system. They include the engineers, programmers, analysts, and manufacturers who build hardware. Evaluation is done by people who ensure that the system is "good." "Good" in the sense it does what it is supposed to do, at a reasonable cost, and is completed in a reasonable amount of time. Evaluation includes the disciplines of Test and Evaluation (T&E), Configuration

Management (CM), Verification and Validation (V&V), and Quality Assurance (QA). Evaluation is also known as product assurance.

Most successful projects will have a balance of each of the three processes. Unfortunately, many project managers practically ignore the evaluation process and emphasize the execution process. This philosophy contributes to the well-publicized expensive and unworkable systems (Ref 9). The APSE is being designed to incorporate tools for all three processes.

In the DOD, the process of building a major weapon system is rarely done in one organization. Four organizations are usually involved in obtaining a weapon system. The project is procured or managed by one Government organization, developed by a private company contracted by the procuring organization, and is used and maintained by two other Government organizations. For example, in the Air Force, a typical alignment is as follows: the procuring organization is Air Force Systems Command (AFSC) who hire McDonnell-Douglas, the user organization is the Tactical Air Command (TAC), and the support organization is Air Force Logistics Command (AFLC). With the exception of the users, all organizations will be involved with the processes of managing the system. But the procuring organization is held responsible for the development of the system and is held accountable for any problems. It is, therefore, especially important that they are well-managed. This paper will emphasize their responsibilities.

System Life-Cycle

From the time a system is conceived until it is no longer needed, it passes through different phases. These phases are known as the system life-cycle. Different authors give different labels to each phase, but the meanings are similar. This paper will use the Government terms explained in the 800 series of regulations. These regulations cover the acquisition of embedded computer systems. Since the APSE is being sponsored by the Government and is being developed primarily for embedded computer systems, this is an appropriate choice.

Before discussing the life-cycle, two terms need to be defined. The first is baseline. A baseline is a document or set of documents that have been accepted and are under CM. It is used as a reference point for major milestones in the system life-cycle. The second term is reviews and audits. Reviews and audits are conducted at various times in the life-cycle. They are attended by representatives from the involved organizations and are presented by the developers. Through the reviews and audits, the baselines and design documents are formally accepted. After a review or audit, the committee can either approve, disapprove, or give contingent approval. In contingent approval, the unsatisfactory parts of the document or product must be changed by the developers. Table I summarizes the reviews and audits that occur during the system life-cycle. The discussion below describes a typical system life-cycle. Not every system will fit into this pattern.

The first phase of the system life-cycle is called the

TABLE I

Overview of Reviews and Audits

Review/Audit	When it is Conducted	Formal Baseline Established	Main Purpose Concerning the Software
SRR System Requirements Review	End of conceptual phase	FB	Establish system requirements Review what system requirements are
SDR System Design Review	End of validation phase	AB	Overview of system requirements CPCIs named Language & facilities needed and tentative schedule & management controls
PDR Preliminary Design Review	Early in full scale development (during analysis phase of software life-cycle)	N/A	Review functional flow and interfaces Review storage needs, data base needs, security, facilities, timing requirements
CDR Critical Design Review	After PDR in full scale development (during design phase of software life-cycle)	N/A	Review that design meets the requirements of the AB Review interfaces
FCA Functional Configuration Audit	End in full scale development (during test and integration phase of software life-cycle)	N/A	Validate that the CPCIs have been completed and meet the requirements of the specs

TABLE I (Continued)

Review/Audit	When it is Conducted	Formal Baseline Established	Main Purpose Concerning the Software
PCA Physical Configuration Audit	After FCA	PB	Verify that the as built CPCI agrees with technical documentation
FQR Formal Qualification Audit	After PCA if it cannot be combined in FCA	N/A	Check system meets all the contract requirements

conceptual phase. During this phase, the basic purpose and requirements of the system are defined. The System Specification is written. This document

> . . . states the technical and mission requirements for a system, . . . defines the interface between these elements and specifies system level requirements (Ref 4:68).

It also includes the general requirements of the software such as the language it will be written in and what standards it will follow. This document will become the Functional Baseline (FB) after it is accepted by the System Requirements Review (SRR).

The second phase of the system life-cycle is called the validation phase. During this phase, the basic concepts of the system will be validated. For development purposes, at this time the system splits into two major parts: software and hardware. Each major software part of the system is called a Computer Program Configuration Item (CPCI). While each major hardware part is called a Configuration Item (CI). For each CPCI and CI, a Development Specification is written. It

> . . . specifies the performance, design, and validation requirements for a CPCI in operational, functional or mathematical terms and in sufficient detail to serve as a contractual definition of both the product to be delivered and the formal process by which it is validated (Ref 4:72).

After they are accepted through the System Design Reviews (SDR), they become the Allocated Baselines (AB).

The third phase of the system life-cycle is the full Scale Engineering Development. During this phase, each CPCI and CI is designed, tested, and built. They are integrated and the whole system

is tested. The software has its own life-cycle. The phases are Analysis, Design, Coding and Checkout, and Test and Integration.

During the analysis, the preliminary Product Specifications are prepared. The Product Specification gives the detailed design of a CPCI as required by the AB and is usually written in three stages. They are:

1. A preliminary version defines functional flow, storage allocation, control functions, and data base structure sufficiently to guide detail design.

2. A complete code-to-version defines the entire design in terms of detailed technical descriptions and flow charts.

3. The final version describes the coded version of the CPCI, includes actual timing and storage values, and includes or references a complete source or object listing (Ref 4:75).

The Preliminary Product Specifications are accepted through the Preliminary Design Reviews (PDR). During the design phase, the Product Specifications are updated to item 2 described above and accepted through the Critical Design Reviews (CDR). During Code and Checkout, the code for each CPCI is written and compiled. During Test and Integration, the CPCIs are integrated and the system is tested. The final Product Specifications are also written and, through a series of reviews and audits (Functional Configuration Audit (FCA), Physical Configuration Audit (PCA), Functional Qualification Review (FQR)), become the Product Baselines (PB). A PB includes the computer listing of the CPCI.

The fourth life-cycle phase is called the Production Phase. During this phase, the system is built and delivered to the user. The last phase in the system life-cycle is the Deployment Phase. During this phase, the system is transported to its operational site, installed, demonstrated, and tested. The Deployment Phase may begin before the Production Phase is completed. If necessary during the Deployment Phase the system is modified or enhanced. The system remains in this phase as long as it is operational.

As a summary, Figure 2.1 shows the system and software life-cycle and the major events that occur during each of them.

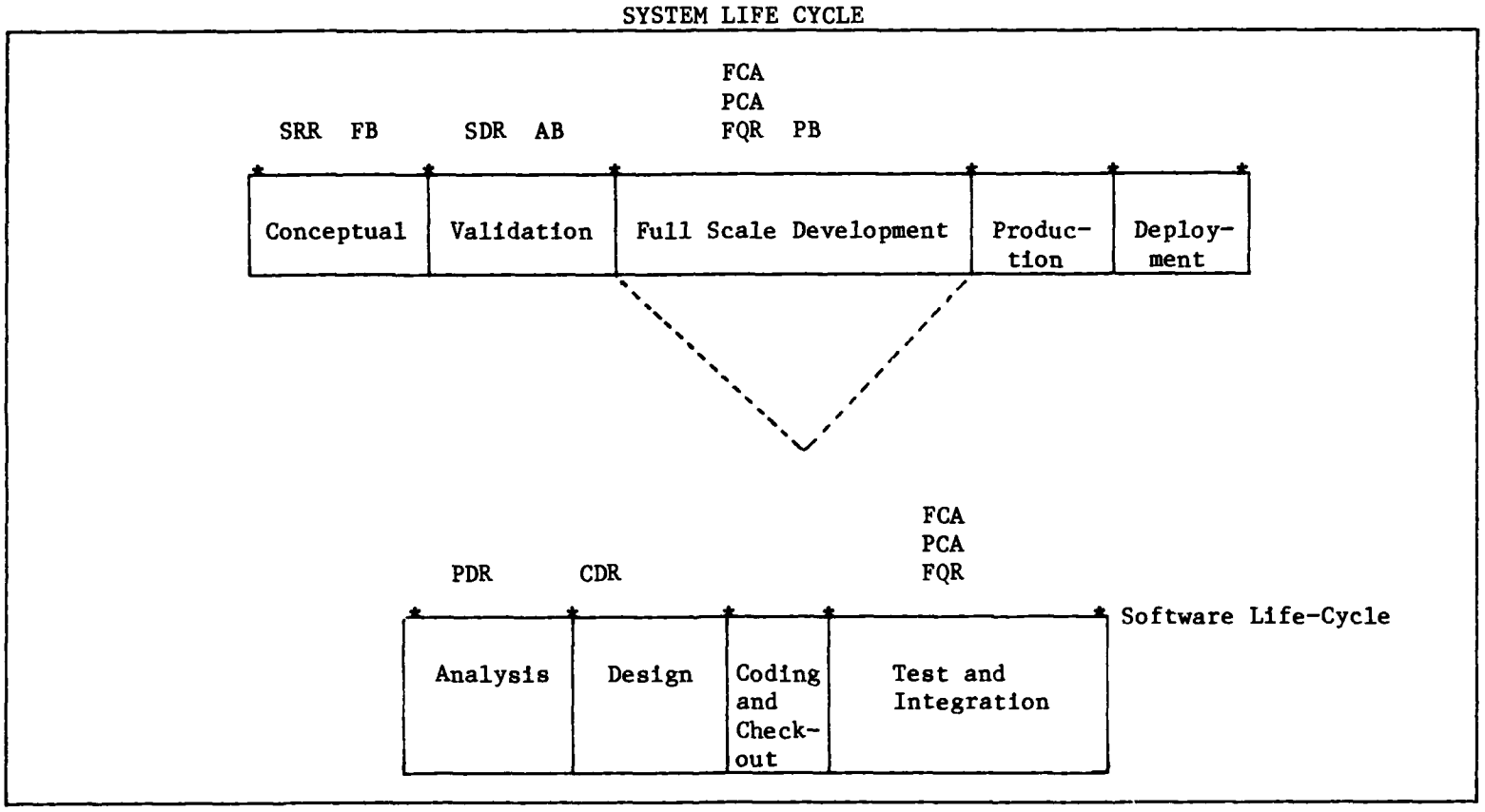


Fig 2.1. System Life-Cycle

III Software Configuration Management

Introduction

SCM is an important part of successful program management. Through SCM management can help reduce the cost of the system, ensure that it is delivered on time, and does what the user expects it to do. Current SCM procedures are a combination of manual and automatic data process'ng techniques. The discipline is nonstandard. Many system development shops have excellent procedures, while all too many have none at all. The success of SCM is very dependent on the knowledge and capabilities of the individual manager. The APSE will help correct this situation by offering an environment that automates many of the SCM tasks and makes the other easier.

SCM is defined as

> . . . the discipline of identifying the configuration of a system at discrete points in time for purposes of systematically controlling changes to this configuration and maintaining the integrity and traceability of this configuration throughout the system life-cycle (Ref 8:20).

As stated earlier, SCM is divided up into four functions. They are Configuration Identification (CI), Configuration Control (CC), Status Accounting (SA), and Configuration Auditing (CA). In this chapter, for each function the following information will be given: the definition, why it is a necessary part of SCM, when it is done in the life-cycle, and how it is currently done.

Configuration Identification

Configuration Identification (CI) is usually defined as the set of technical documents that define the functional and physical characteristics of a system. Government documents specify three documents that are generally used. They are the Functional Configuration Identification (FCI), the Allocated Configuration Identification (ACI), and the Product Configuration Identification (PCI). These documents are also known as the to-be-established baselines. As stated in Chapter II, the FB is a system specification, while the AB and PB are development and product specifications. CI also includes the process of identifying and labeling the software parts of the system.

Each major software part of a system is called a Computer Program Configuration Item (CPCI). All the CPCIs together form the software part of the system. The CPCI is the most important component of the software items because it is controlled by SCM. Each CPCI goes through the software life-cycle. Therefore, each has its own allocated and product baseline. A change cannot be made to the requirements or design of a CPCI without formal approval. Regular status reports on each CPCI must be issued by the developers and each CPCI will undergo formal test procedures.

Besides the CPCI, other levels in a system are designated. From highest to lowest, the hierarchy is as follows:

System - Includes everything needed to complete the task required. The system includes both inanimate objects and personnel.

System Segment - It is part of a system sometimes referenced as a subsystem or functional area. It may contain more than one functional area and consists of CIs or CPCIs or both. It is used when:

1. A system is purchased incrementally.

2. A part of an existing system needs revision.

3. A system is divided up for different programming offices.

CPCI - "An aggregation of computer programs that satisfies an end-use function and is designated by the Government for CM" (Ref 2:52).

Computer Program Component (CPC) - A functional or logical part of a complex CPCI. It is used for ease in explaining the design of a complex CPCI.

Routine - A subroutine of a computer program. Large and complex systems may require more levels. Small systems may not use them all. When a system is broken down to the routine level, a tree chart can be drawn to show the structure of the system. An example is given in Figure 3.1.

CI is a necessary part of SCM because it helps management control the software. First, through the baselines, the state of the software can be defined at any time. Second, labeling gives a point of reference to everyone involved in the project.

The selection of the CPCIs for a system is not a trivial process and will vary from project to project. If a great amount of control is needed, then more CPCIs will be created. A large system that deals with life and death situations will need more control than a small

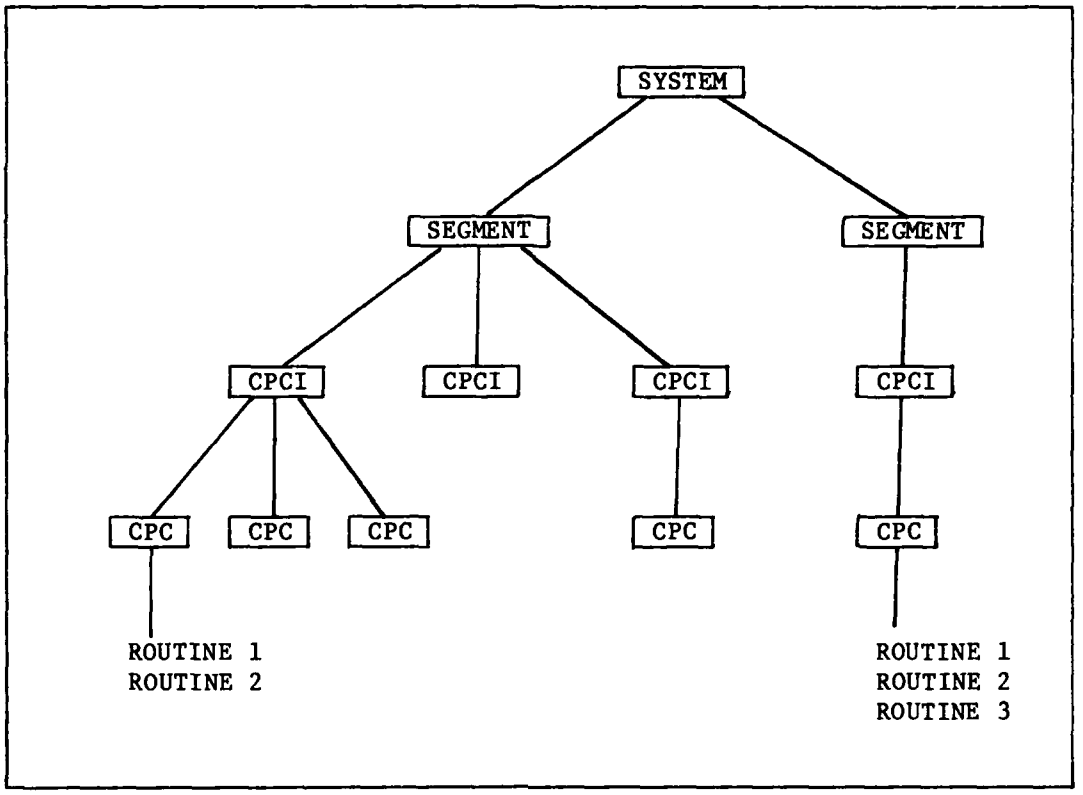


Fig 3.1. Structure of a System Given in a Tree Chart

routine project. In determining the number of CPCIs, trade-offs have to be made. Too many CPCIs and the SCM task becomes too complicated and unworkable. But if too few exist, there will not be enough control and the SCM goals will not be accomplished. The solution is usually to have different size CPCIs in the system. Small CPCIs, such as a single routine, will be chosen for critical areas. Large CPCIs, such as a functional area consisting of many modules, will be used for more routine areas.

The ASD, Airborne Systems Software Acquisition Engineering Guidebook for Configuration Management ASD-TR-79-5024 (Ref 2), gives

some guidelines in selecting CPCIs. First, one must identify the software processes that are needed to use and maintain the system over the entire life-cycle. The processes will include operational, support, and test software. Secondly, one must group these processes into CPCIs. For example, Table II lists some of the guidelines used to choose CPCIs by ASD. The CPCIs of a system will be suggested during the conceptual phase and formalized in the design phase. The selection of CPCIs is not generally done by the SCM people. It is usually done by the software engineers. The SCM people will check to ensure the CPCIs are broken into logical parts and are complete. The CPCIs are the basis for SCM.

Identifiers

After the CPCIs and other components are selected, they and their associated documents must be labeled with identifiers. The system cannot be controlled unless its components have names.

Different kinds of labeling systems exist. By Government standards, an acceptable method has identifiers that have system unique names for each component. The software component must be able to be filed and retrieved on a computer system. Therefore, the identifier must have at least a portion that does not change. Other favorable features for identifiers are variability, traceability, functional significance, pronounceability, and compactness. Variability means that the identifier has a portion that can change to reflect changes, such as new versions. Traceability means that the identifier tells where the component came from. It could identify the

TABLE II

Characteristics Used to Group Software Processes Into CPCIs

Guidelines for Grouping Software
• Of the same type such as operational, test, or support.
• To be used on the same computer.
• Able to be developed and tested by one contractor.
• Needed at the same time in the life-cycle.
• Of the same importance to the system.
• Of the same difficulty to develop.
• In need of the same level of developmental control.
• Small enough to be monitored by one person.

contractor or the position in the structure tree. An example given in Reference 2 is a four-character identifier where the first character indicates the CPCI, the second the CPC, the third the module, and the fourth the routine. Functional significance means that the identifier indicates the function of the component, such as SINE being an identifier for a routine that determines the sine. Pronounceability is a good feature because a pronounceable identifier is easier to remember. The last feature, compactness, means that there must be a limit to the length of an identifier. The restrictions may come from the operating system, Government standards, contractor standards, or administrative needs (not too cumbersome to use). No one method of identifying software components will be able to have every feature mentioned above.

The identifying and labeling of the CPCIs is done in the conceptual phase of the system life-cycle. They are accepted in the validation phase through the SDR and documented in the Allocated Baselines.

Configuration Control

Configuration Control (CC) is the process of controlling the changes made to the software or the documents in the system. CC is involved only for items formally under SCM. For example, a change to a sanctioned baseline, a test procedure, or a technical manual are subjected to CC. Changes result from new requirements or errors found in the system. CC is an important aspect of SCM. Through its methods, whenever a change is made to the system, everyone involved in

the system is aware of it. No one can make a change independently to any information under SCM. Everyone is aware of the latest version of the system. All of the documents reflect the current software. CC is done in all phases after the FB is established.

CC is accomplished through three methods. They are documentation, procedures, and organizational bodies.

The main organizational body for CC is the Configuration Control Board (CCB). The CCB is a group of people involved in the system who make decisions concerning the project. The CCB will approved the changes made to the system. If contractors are involved in the development of the system, two CCBs will probably be formed, the procuring CCB and the contractor CCB. The procuring CCB is set up during the validation phase. Their responsibility will be to approve changes to the baselines. The Program Manager (PM) is usually the chairman of the procuring CCB. Other members include the top managers in each functional area, and if it is a Government procuring CCB, representatives from participating Government agencies. Specialists and contractors may be invited to attend procuring CCB meetings as advisors. The contractor CCB is similar to the procuring CCB, except the members are from their firm and are concerned with the items they are developing. They will be mainly concerned with technical documents.

The CCB is responsible for both hardware and software. If the system being developed is large, a separate CCB for software and hardware may be provided. If the system is being developed for a

third party, a CCB will have to be set up in that organization to monitor changes made to the system after it is operational.

The second method of accomplishing CC is through procedures. Procedures are the methods approved by the CCB to make a change to the system. Procedures are not firmly set. It must be remembered that each CM plan is tailored to each project. SCM is not suppose to encumber the project, but rather improve the quality of it.

The final method of CC is documentation. Forms are the most common way to document. Many forms are used by the Government and private firms. A few of the most common will be discussed. The Change Request (CR) is a form used to state that a change in the system requirements is desired. If a user or auditor discovers a deficiency in the system, he submits a different form called the Software Incident Report (SIR). Both the CR and SIR are analyzed by the developers. After the analysis, they decide if the change is really needed and/or beneficial. If it is, an Engineering Change Proposal (ECP) is submitted to the CCB. The CCB determines if the change will be rejected or accepted. If it is accepted, portions of the life-cycle will be repeated. Rejected ECPs are filed for further reference. Figure 3.2 shows these procedures. Appendix A gives an example of a CR, SIR, and ECP.

CM Plan

Although not a direct function of CC, the CM plan is discussed here because it is a controlling activity. The general plan should be formulated during the conceptual phase. If the system consists of

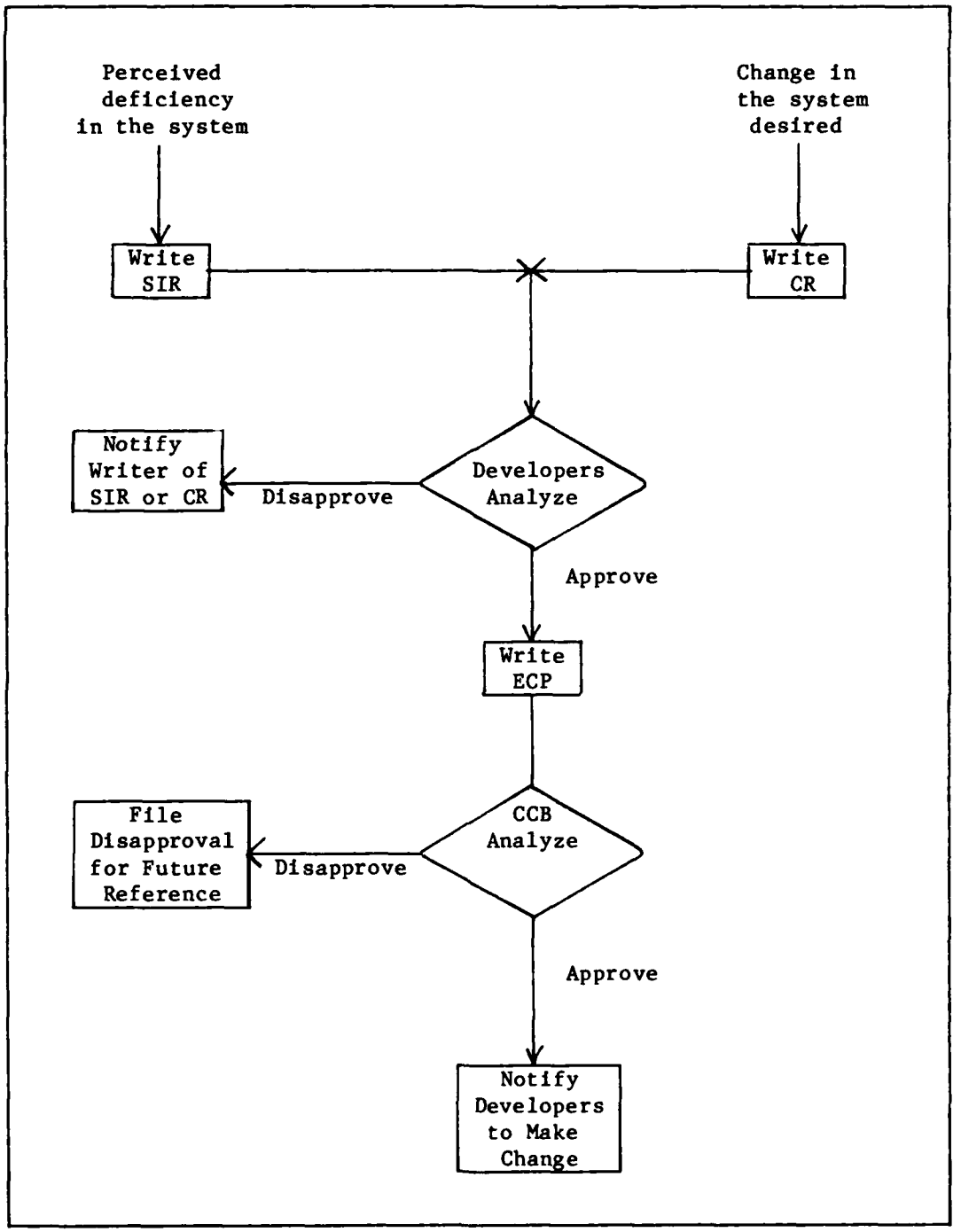


Fig 3.2. Typical Steps Used to Make a Change to the System

both hardware and software, separate plans can be made early in the full scale development phase. According to Reference 8, topics that should be covered in the CM plan area:

1. An overview of the system.

2. CM organization - Besides the four functions, the role of the CCB should be discussed.

3. CM tools - This part should describe the forms that will be used, labeling conventions, and any automated aids.

4. CM procedures - This part will state what will be done during each stage of the life cycle.

5. CM resources - At this time, the money and staff needed will be given.

In the Government, the CM plan may be given in other required documents. These documents are the Program Management Plan (PMP), Computer Resources Integrated Support Plan (CRISP), Statement of Work (SOW), and the Contract Data Requirements List (CDRL). The PMP is written by the Program Office early in the Conceptual phase. It is a directive for everyone involved in the system on the overall acquisition plan. It is changed whenever necessary. The CM plan is developed following this plan and, when approved, may become part of it. The CRISP lists the computer resources needed throughout the system life-cycle, including CM resources. The SOW is a document that defines the developers (contractors) tasks to complete the project. The CDRL lists all the deliverable contract items. Each item in the CDRL has a Data Item Description (DID) number. The DID refers to a

Government accepted outline for the specified document. Appendix B gives a DID for a Software Configuration Management Plan.

Status Accounting

Status Accounting (SA) is a process of recording, storing, and reporting the history of a project. The history tells when, how, and why the events happened. The history is important for many reasons. First, it can help new personnel learn about the project. This is especially important if they join in the middle or end of a project. Studying the history of a project can also help people learn about developing software. From studying past projects, they can try to repeat the good features and avoid the mistakes. Also, cost estimates, staff needs, and time schedules can be made from studying past projects. Lastly, a log of the history can be used for preventative purposes. From past experience, areas that have been shown to cause problems can be corrected before the problems develop. SA is done during all phases of the life-cycle. SA's importance increases as the life of the project increases. Frequent personnel changes are a common problem in the military environment. During a project that has a long life span, generally more people will come and go. SA will help them become part of the project. Also, SA is more difficult when the project is very complex. The process of storing and recording what happened and why will be harder. An effective SA plan is a major factor in successful SCM.

As stated before, SA involves recording, storing, and reporting data from the projects. Exactly what should be recorded depends on

the size and complexity of the project. It is better to record too much than too little. Bersoff, Henderson, and Siegel (Ref 8) suggest, as a minimum, it is necessary to record the events required to sanction a baseline and the events following a CR and SIR. Figure 3.3 gives the typical events recorded following a CR. Some projects will need to have a detailed description of what, why, when, and how for each event, while other may just need a what and when. What is required should be stated in the CM plan. This data is stored in files. The storing and recording will be done manually or automatically in a Data Base Management System (DBMS). For Government agencies, DODD 5010.19 states that

> . . . automation of status accounting shall be employed only when the volume of data or rapid response time makes it necessary and it is economically feasible (Ref 2:112).

With either method, the life-cycle of each CPCI should be traceable.

The last function of SA, data reporting, is the method used to keep the project personnel informed about the project. Bersoff, Henderson, and Siegel (Ref 8) suggest the following reports be made. They are:

1. CCB, Review or Audit minutes.
2. Periodic baseline status reports.
3. CR and SIR status reports.
4. Executive summary of SCM activities for management review.
5. Baseline releases.
6. Ad hoc reports generated by request.

Government documents give specific names to the documents that will be

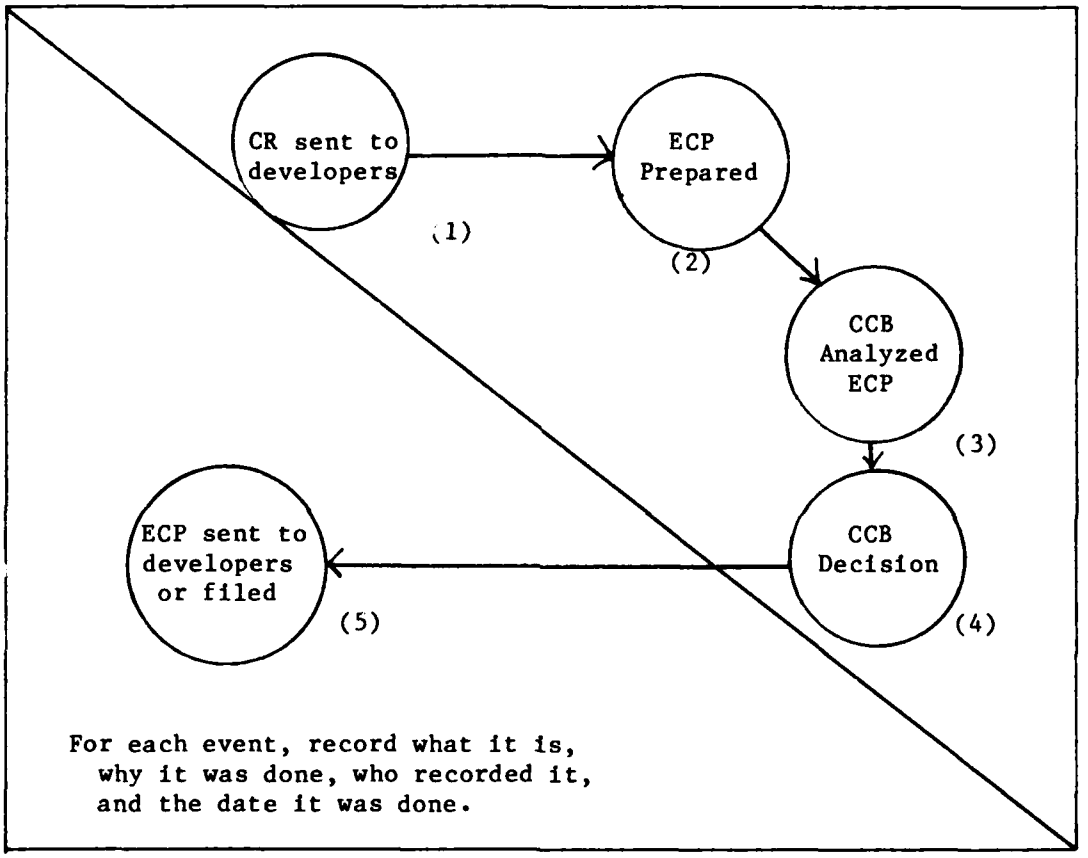


Fig 3.3. Typical Events Recorded for a CR

produced from the SA activity, but the information is basically the same as the above list. All reports that are required should be stated in the CM plan.

Configuration Auditing

The last function of SCM is Configuration Auditing (CA). Government documents define CA as the FQR, FCA, and PCA. These audits are designed primarily to ensure that the software parts of the system (including documents) do what they were designed to do. As stated in Chapter II, these audits are performed near the end of the Full-Scale Engineering Development phase. They are first performed at the CPCI level and then at the system level. If an error is discovered, the auditor must go through the formal CC procedures to implement a change.

Bersoff, Henderson, and Siegel (Ref 8) define CA in a different manner. Their definition states that audits are done throughout the life-cycle and preferably done by independent auditors. The Government considers this to be part of the Verification and Validation (V&V) process. Table III gives an overview of what V&V is and is not (Ref 7).

The Government recommends V&V be done only when it is "economically justified in terms of life-cycle benefits" (Ref 7:19). The following examples were given as projects that would justify V&V:

1. Software with a high cost of failure (e.g., space systems).

2. Software for which the cost of error detection through operational use is greater than the cost of audits (e.g., aircraft operational flight programs).

TABLE III

Verification and Validation (V&V) Characteristics

Independent Verification And Validation Is	Is Not
An independent technical activity	Conducted by the personnel that develop the software
Aimed at product evaluation throughout the life-cycle	Checking the code during Development Test and Evaluation (DT&E)
Identifying errors early	Identifying errors during DT&E
Employed to ensure that all system and subsystem requirements have been fulfilled by the software	Employed to ensure that only test requirements of the computer program development specification are met
Complementary to the development effort	A duplication of development activities
Designed to help the developers	Conducted to harass the developer
Additional insurance	A guarantee of success

3. Real-time software which must work under all scenarios (e.g., nuclear safety programs).

Many people disagree with the Government and feel, despite the high cost, V&V should always be done. The money spent up front will pay in the long run by producing a better product. The advantages include:

1. Improved reliability - Fewer errors are found after the system is operational.

2. Greater visibility - The chance of success is increased.

3. Reduces the cost - Errors are found earlier in the life-cycle when they are easier and cheaper to correct.

Accomplishing any kind of audit is a complex process that needs to be managed by experienced people. It is done using many techniques and tools. Table IV gives a list of some of the tools and techniques. Explanation of these is beyond the scope of this paper.

TABLE IV

List of Tools and Techniques

Tools		
Accuracy Study Analyzer	Dynamic Analyzer	Relocatable Loader
Assembler	Dynamic Simulator	Requirements Language Processor
Automated Test Generator	Editor	Requirements Tracer
Comparator	Engineering (Scientific Simulations)	Restructuring Program
Compiler	Environmental Simulator	Software Monitor
Compiler Validation System	Flowcharter	Standards Enforcer
Consistency Checker	Hardware Monitor	Statement-Level Simulator
Cross-Assembler	Instruction-Level Simulator	Static Analyzer
Cross-Reference Program	Instruction Trace	Test Beds
Data Analyzer	Interface Checker	Test Drivers, Scripts, Generator
Decision Tables	Interrupt Analyzer	Test-Result Processor
Decompiler	Logic/Equation Generator	Timing Analyzer
Design Language Processor	Overlay Program	Trace
Diagnostic/Debug Aids	Path Analyzer	Units Consistency Analyzer
Driver	Program Sequencer	Workload Analysis Aids

TABLE IV (Continued)

Techniques		
Algorithm Evaluation Test	Flight Tests	Prototyping
Analytical Modeling	Functional Testing	Simulation
Capability Matrices	Logical Testing	Standardization
Code Inspection	Modular Programming	Static Analysis
Correctness Proofs	Path Testing	Stress Testing
Design Inspection	Performance Evaluation	Structured Programming
Emulation	Post-Functional Analysis	Symbolic Execution
Equivalence Classes	Process Construction	System Simulations
Error-Prone Analysis	Production Libraries	Top-Down Programming
Execution Analysis		Walk-Thrus

Summary

SCM is an involved procedure that involves many techniques. Software Configuration managers deal with both the management and the doers. They watch over the project to ensure a quality software product is being produced. They have been described as the policemen of the project. As a review of the SCM process, Figure 3.4 gives the major SCM milestones that happen in each phase of the life-cycle. It should be noted that these milestones are for the "average" Government-sponsored embedded computer system. SCM for smaller software projects may not include all of the milestones.

Conceptual Phase	Validation Phase	Full Scale Engineering Development Phase				Production / Deployment
		Analysis	Design	Coding and Checkout	Testing and Integration	
•System Specification written •SRR to establish FB *CCD Set up *Begin CM Plan	△*CC, CA, and SA procedures implemented •CPCIs selected •Development Specification written •SDR to establist AB	*Complete CM Plan •PDR to establish Preliminary Product Specification	•CPCIs identified •CDR to establish second Preliminary Product Specification	*Ensure CC is done	*Ensure CC, △SA done •FCA, PCA, □FQR to establish PB □FCA, PCA FQR to verify & validate system & related documents • - Process part of CI * - Process part of CC △ - Process part of SA □ - Process part of CA	*Ensure procedures exist for SCM for maintenance of system.

Fig 3.4. Major SCM Milestones In the System Life-Cycle

IV Requirements

Introduction

This chapter will present the requirements of the Ada Programming Support Environment (APSE) SCM tool. In addition, the specific requirements of the SA part of the tool are given. Prior to this, an overview of the APSE is presented.

Ada Programming Support Environment (APSE)

As stated previously, the APSE is a DOD-sponsored project to create an environment to produce software primarily for embedded computer systems. A major feature of the APSE is the new computer language Ada. Ada is designed to be the single computer language used in DOD-developed embedded computer systems. Not only can Ada handle the special requirements of embedded computer systems, but it offers all the conventional capabilities of a general purpose language. An important feature of the Ada design is that it emphasizes program reliability and maintenance. This is achieved by choosing a design that stresses program readability over ease of writing code.

Although Ada is an important feature, the APSE is more than a new computer language. As the name suggests, it is an environment. The philosophy of the APSE design is explained in "Stoneman." The design will be based on a few simple concepts which are ". . . straight forward to use and understand" (Ref 10:14). The APSE

will support all project members throughout the entire life-cycle of the project with a complete set of the integrated tools. All the tools and user programs will be written in Ada. They will be machine independent. As much as possible, the APSE itself will be portable. Finally, the APSE will be a dynamic system which can always be improved upon.

The APSE will be composed of three basic parts: the data base, the interface (user and system), and the tool set. The data base will hold all the information concerning a project during its entire life-cycle and will be a key feature of the APSE. The data base will store uniquely named objects. Objects, as defined by "Stoneman," are ". . . identifiable collections of information" (Ref 10:18). Typical objects would be a test data file, a documentation file, or an Ada source file. Different versions of each object can exist. Different groups of objects can be put together to form "software configurations." A configuration is itself an object; thus, different versions of it can exist. Large groups of objects, such as all objects pertaining to a project, can be grouped together into partitions. Access controls can be used on both the object and partition level.

The data base will permit relationships between objects. The user will be able to travel through the networks formed by these relationships. Each object will be supplemented with a minimum of history, categorization, and access attributes. The user will be able to access both the information in the object and the attributes. The

history attribute contains all the information necessary to maintain a complete history of the object. The history attributes will be the basis for configuration control. The categorization attribute will contain the category of information contained in the object. This information will be used to protect the integrity of the object by indicating which parts of the object cannot be changed by any operation. The access attribute contains which executing programs and users have access rights to the object. In addition, the data base will be able to generate reports and system statistics.

The second part of the system is the interface. The user interface will allow the user to use the APSE tools through a command language. The command language will be machine independent. The system interface will allow intercommunication among the APSE tools.

The last major part of the APSE is the tool set. It will provide a complete set of integrated tools for Ada program development, maintenance, and configuration control. Like other parts of the APSE, the tools are portable, written in Ada, user-friendly, and open-ended. The other requirements will be discussed in the next section. Some examples of APSE tools are the compiler, editor, debugger, and linker.

In order to achieve the goal of machine independence for both the user programs and tool set, "Stoneman" defines two lower levels in the architecture of the APSE: the Kernal Ada Program Support Environment (KAPSE) and the Minimal Ada Program Support Environment (MAPSE). To explain these levels, "Stoneman" gives the diagram shown

in Figure 4.1. Level 0 is the only level that is machine dependent. It contains the host hardware and software as needed. This part is kept as small as possible in order that it can be easily modified to fit any machine, allowing the APSE to be as portable as possible. The KAPSE is a key feature of the APSE. It is similar to an operating system. It contains the program execution facilities, the data base, the data base management system (DBMS), and the interfaces needed. "Stoneman" does not require the KAPSE to be written in Ada if it has to make use of the local operating system, filing system, or DBMS. APSE portability would be lessened if it is not written in Ada. The MAPSE is a smaller version of the APSE. It contains the minimal set of tools needed to run an Ada program. The specific tools are listed in Figure 4.1. The MAPSE is written in Ada. The APSE is an extension to the MAPSE.

SCM Manager Requirements

"Stoneman" requires that the APSE includes a configuration control system as part of the tool set. The guidance given in "Stoneman" is of a very general nature. Specifically, it states

> The history attributes provided at the KAPSE level record a variety of software configuration relationships. Tools to help structure these relationships, modify them, indicate the ramification of (potential) modifications, etc., are appropriate in an APSE. In many systems, the facility will be provided, subject to suitable controls, to archive or delete superseded material in the data base or to rederive material subsequent to and affected by changes (Ref 10:42).

The actual design and how it will be implemented is left open.

Since the software configuration manager is part of the APSE

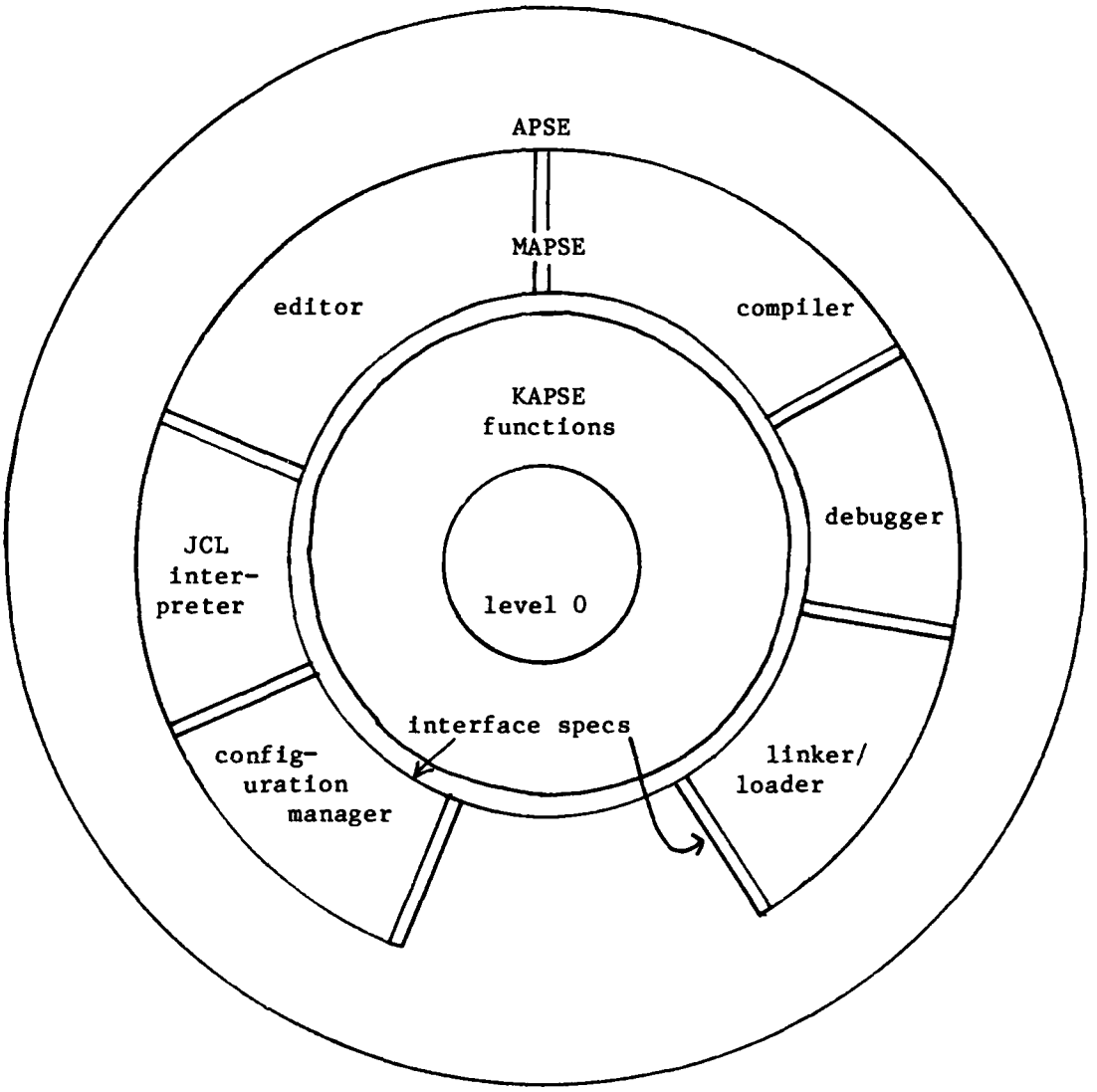


Fig 4.1. APSE Structure (Ref 10:2)

tool set, it must meet all the requirements set forth in "Stoneman" for an APSE tool. These requirements are explained below:

1. An APSE tool must be designed to meet a clear functional need. In this case, the tool must automate as many of the SCM tasks as possible.

2. An APSE tool must be written in Ada.

3. If possible, an APSE tool will be designed to conform to standard interface specifications.

4. An APSE tool will be machine independent and portable.

5. An APSE tool will be designed to be open-ended. It will always be able to be improved upon.

6. An APSE tool will be user-friendly. Help messages will be offered to the user. In the case of the SC manager, it must be designed primarily for use by managers and administrators. People who may have very little experience using automated tools.

7. An APSE tool must be reliable.

8. When necessary, an APSE tool must communicate with other APSE tools. This requirement is especially important to the software configuration manager. For example, the software configuration manager will communicate with the documentation system because documentation is an essential part of SCM.

9. Communication between the user and an APSE tool will be done through uniform protocol conventions.

10. When necessary, an APSE tool will generate reports.

Besides meeting the requirements of an APSE tool, the software configuration manager has requirements arising from the definition of SCM. They are:

1. The SC manager will assist the user in the preparation of documents. Preparing documents is an important part of SCM. Unfortunately, this task is often considered the most tedious part of SCM. The automated tools of both the APSE and the SC manager will make this task less tedious.

2. The SC manager must allow the user to easily check on the status of any object. The tool will obtain this information from the object's history attribute. One of the main reasons for SCM is to control the software throughout the lifecycle. Knowing the status of each part of the system will help control the software. Areas that are causing problems can be identified earlier when they are easier to correct. This requirement is a key to any APSE SC manager.

3. Since the history attribute is so important, the tool must assure that it is filled out each time an object is created or modified.

4. The tool will ensure that the ability to trace the development of the software system exists. If necessary, a previous version of the system will be able to be recreated. This ability is required by the definition of SCM.

5. Standard forms are an integral part of SCM. The tool must be able to generate them and assist the user in filling them out. If

the required information is in the system, the user should not have to supply it again (i.e., user's name, project name, date, etc.).

6. The tool will assist the user in disseminating information electronically to any other project member. This will make the task of SCM easier by ending the need to make copies of documents and manually delivering them.

7. The tool will generate any statistics that are beneficial to the project.

As can be seen, the APSE software configuration manager will be composed of many different parts. In order to avoid redundancy, the tool will need to share features from other parts of the APSE.

SA Requirements

The SA part of the APSE software configuration must fulfill the requirements explained above. The goal of SA is to maintain a complete history of a project. This is accomplished by recording the information pertinent to the project, storing it, and when necessary publishing it.

The SA part of the APSE software configuration manager must meet all the requirements of an APSE tool.

1. The SA part of the tool will meet a need. It will record information, store it in the data base, and when necessary generate reports.

2. When possible, the SA part of the tool will use standard interface conventions.

3. All the SA software will be written in Ada.

4. The SA tool will be machine independent and portable.

5. The tool will be designed so that it can be improved upon.

6. The design to the tool will be based on concepts that ensure the tool will be reliable.

7. The tool will be user-friendly. The queries into the data base and any other instruction issued by the system will be easy to learn and use. Clear "help" messages will help the users when an error occurs.

8. The SA tool will communicate with other APSE tools. It will use the documentation system for preparing documentation, the filing system to keep track of how long objects will remain in the data base before being archived, and the DBMS for queries into the data base.

9. The SA tool will be designed to use uniform protocol conventions.

10. The SA tool, upon command, will generate reports. Reports to be generated include status reports, CCB minutes, and review minutes.

As with the configuration manager, the SA part of this tool has other requirements besides those of an APSE tool. These requirements are required to fulfill the definition of SA.

1. Documentation is a major part of the SA function. The SA tool will ensure that the preparer is assisted in preparing all documentation. Documentation will not be written by the machine, but the machine can make the task less tedious.

2. The SA tool needs to access the history attributes of the objects. This is necessary to obtain the information on the status of the project and trace the history.

3. By definition, the SA tool needs to be able to trace the history of a project. This will be done by being able to recreate previous versions of the project.

4. In order to maintain a history of a project, one must keep track of the changes made to it. Forms are used to keep track of changes. The SA tool will need to ensure the proper forms are generated. To assist the user, the forms needed in a project will be partially filled out by the system if the system already has the information. Name, date, and project number are examples of information the system should be able to supply.

5. The SA tool will maintain any statistics the project manager determines necessary.

6. The SA tool will be a flexible tool. It will allow the amount of control over the software to vary from project to project.

V Functional Design

Introduction

In this chapter the functional model of the SCM tool is presented. It is presented using Structured Analysis and Design Technique (SADT). SADT is a method designed by SOFTECH to perform functional analysis and design. In this paper only the functional analysis or what the system is supposed to do will be presented. The design or how it will be implemented is left for further research.

SADT is a way to diagram in a top-down, modular, and structured way the model of a system. The complete model consists of a set of diagrams or nodes. The first node A-0 (read A minus zero) is the most general. It consists of only one part or box that specifies the general function of the system. The following nodes get more detailed. Each one consists of three to six boxes. Each box represents an activity performed. The boxes are connected by arrows. These arrows show how the boxes interface with each other. SADT does not show sequence like a flow chart. The meaning of the arrows are shown in Fig 5.1. The items indicated by the input arrows are transformed by the activity marked in the box to the items marked by the output arrows. The items represented by the control arrows govern how the activity is done. The mechanism arrow is less frequently used. It shows the device which performs the activity. This

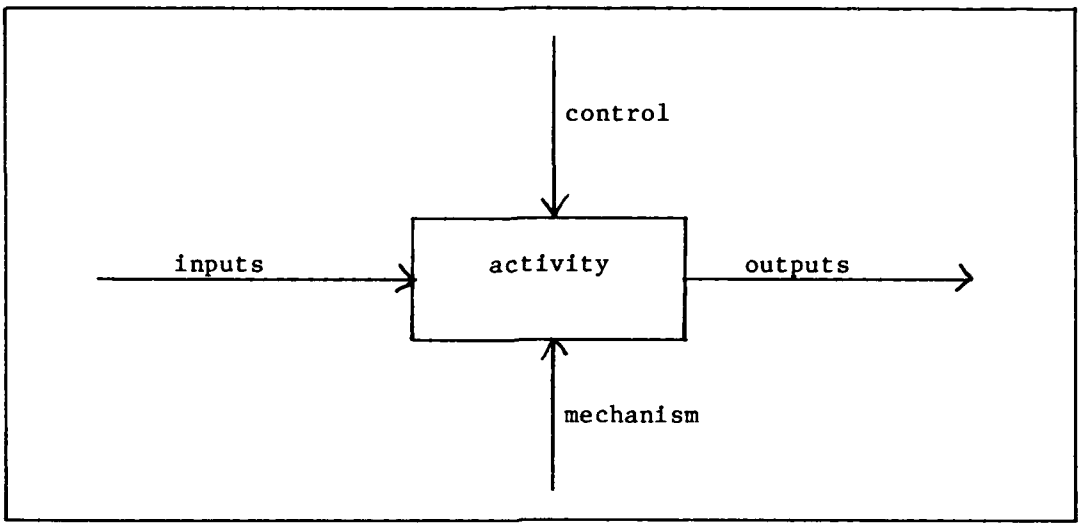


Fig 5.1. SADT Arrow Definition

discussion is a very simplified view of SADT. For a more complete discussion, see reference 23.

Diagrams

Node Index and Corresponding

Decomposition Structure of Diagram

A-0 Accomplish SCM in the APSE (context)

A 0 Accomplish SCM in the APSE

A 1 Perform CI

A 11 Prepare FB

A 12 Identify and Label CPCIs

A 13 Prepare AB

A 14 Prepare PB

A 144 Do PB Audits

A 2 Perform Configuration Control

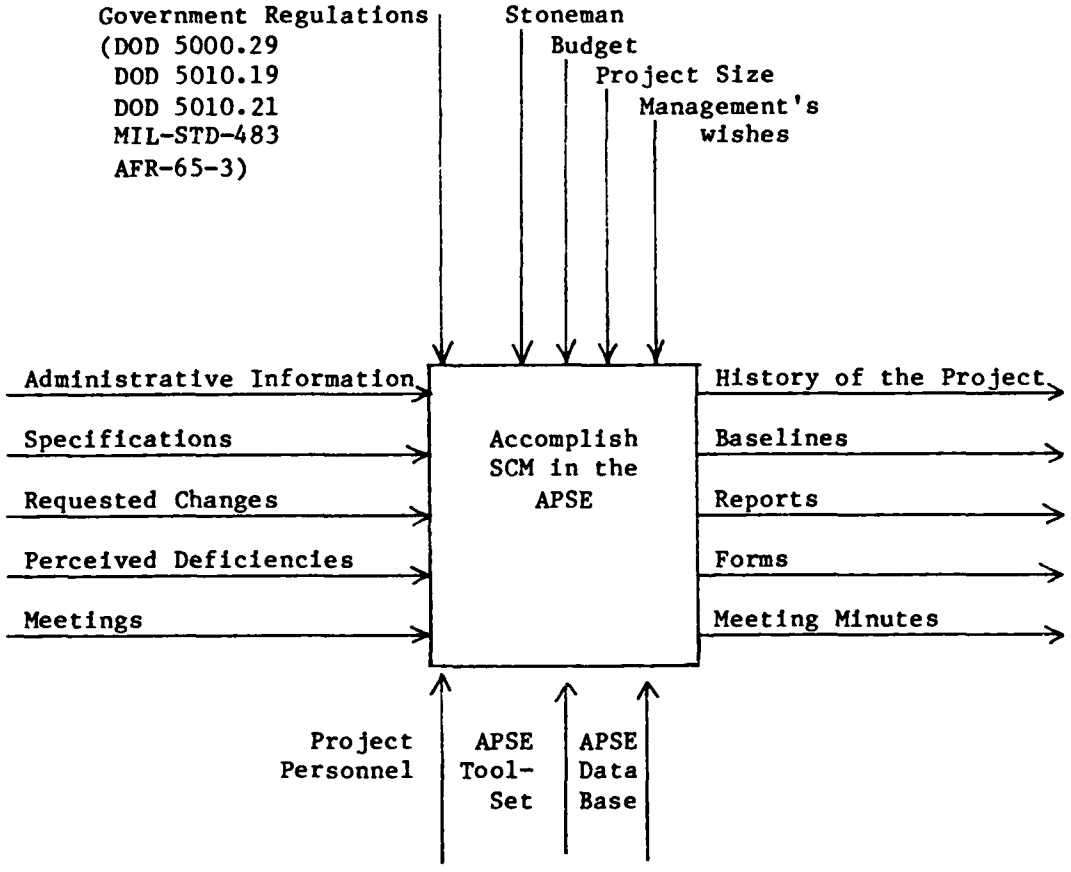
A 3 Perform Status Accounting

A 31 Record Events

A 32 Store Events

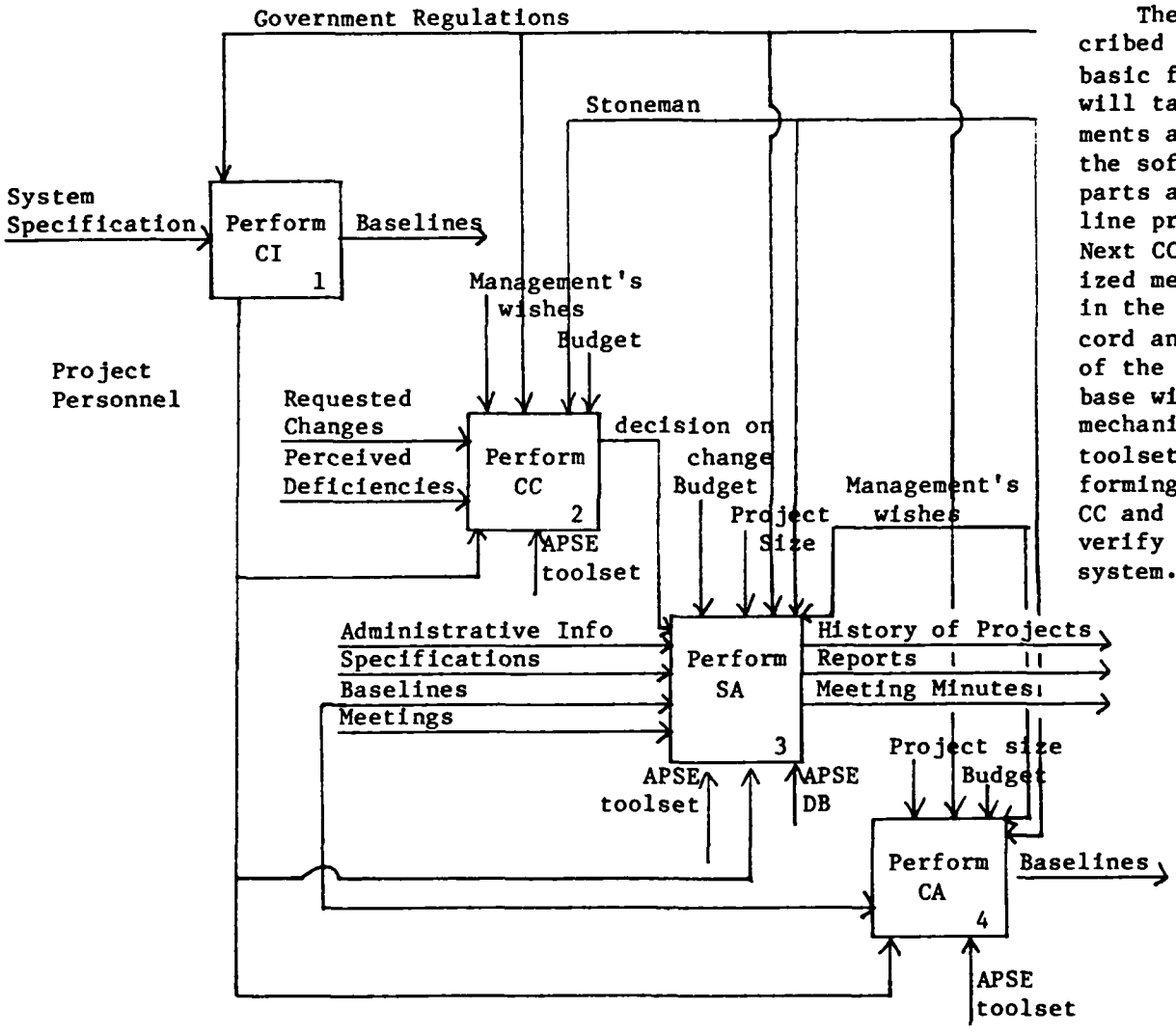
A 33 Report Events

A 4 Perform CA



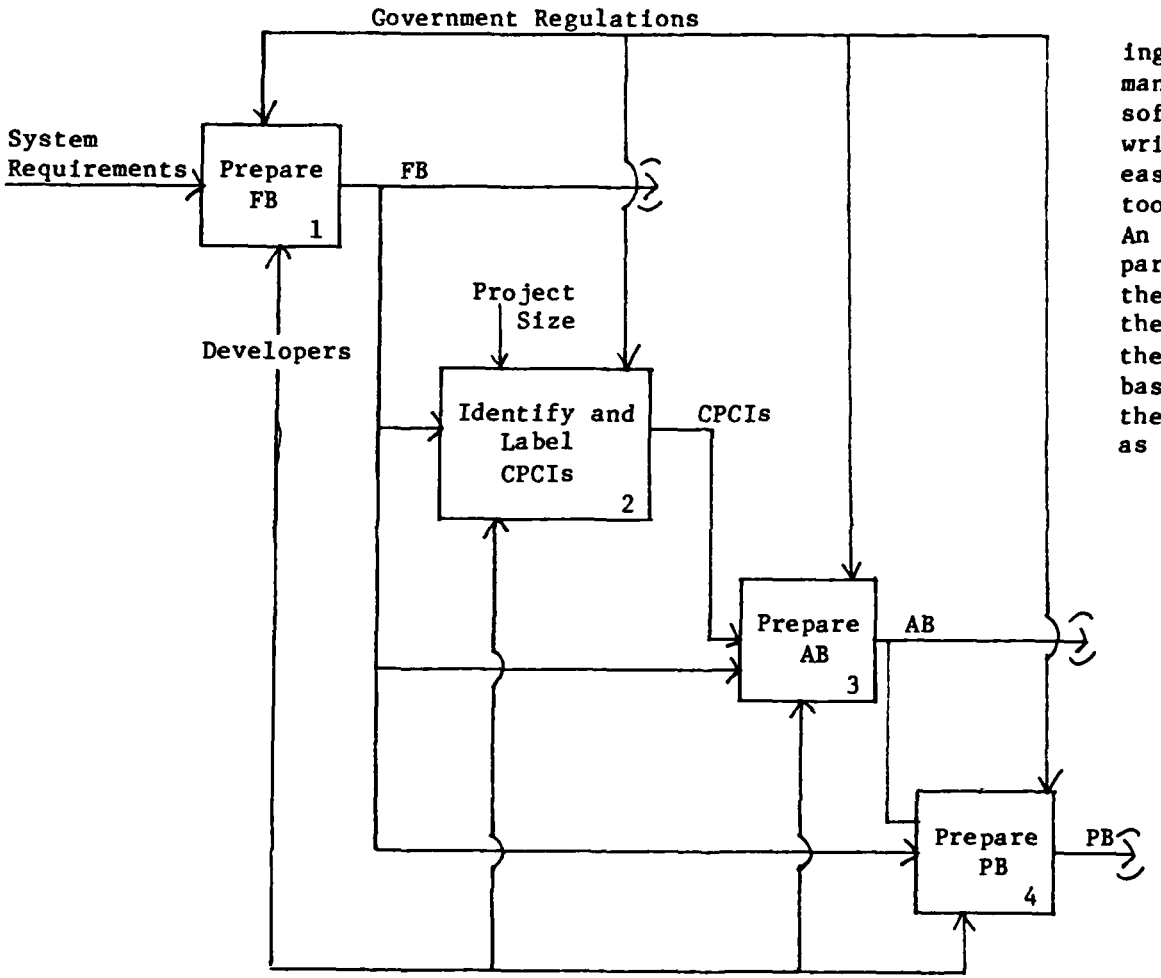
NODE A-0 TITLE - Accomplish SCM in the APSE

A-0 is the highest level diagram. It describes the general function that must be accomplished. In addition, the inputs, outputs, controlling factors, and the mechanisms are shown. As can be seen, a wealth of data will be stored in the data base and must be managed. The administrative information includes who is in the project, what their responsibilities are, and when suspense items are due. Specifications include the System, Development, and Product Specifications. These imply the CPCIs. From the inputs the outputs are produced. They include the Functional, Allocated, and Product Baselines. Reports are another input. Reports that indicate the status of changes, baselines, administrative tasks, and other documents such as user manuals will be issued. Government regulations, Stoneman, the budget, project size, and management's wishes will control how SCM is done in the APSE. The tool will allow the amount of control to vary from project to project.



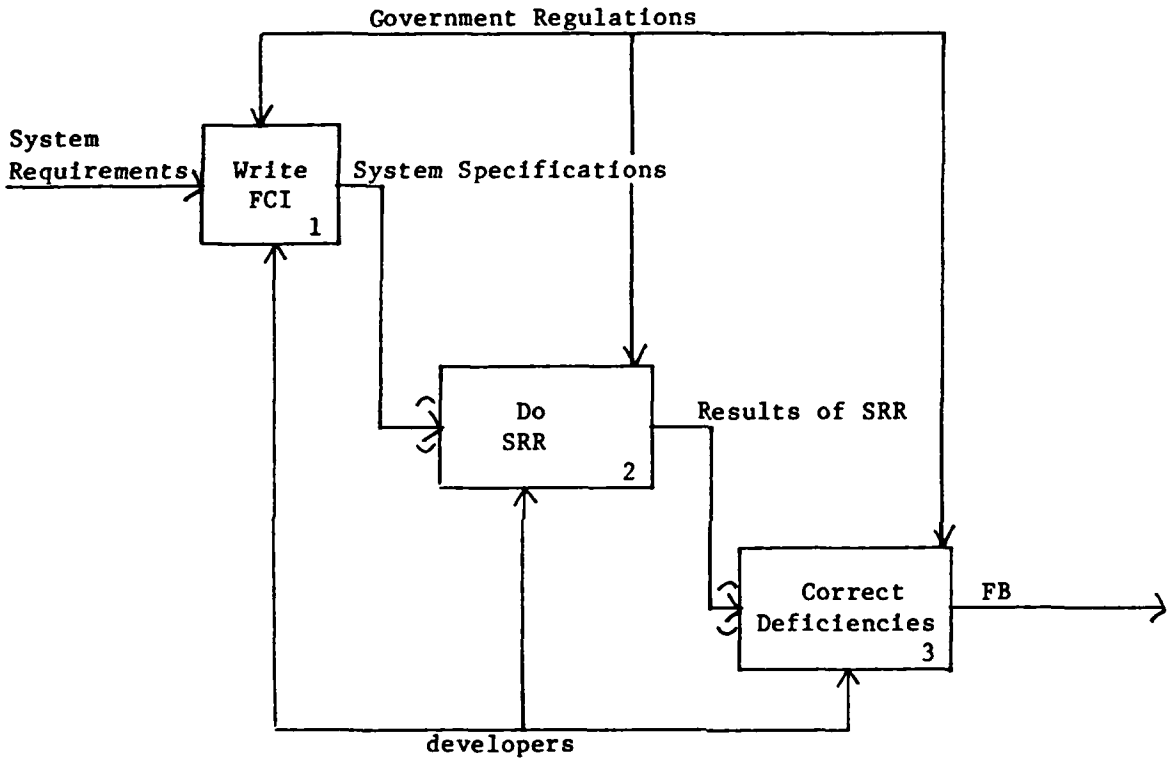
NODE A 0 TITLE - Accomplish SCM in the APSE

The four activities described in A 0 are the four basic functions of SCM. CI will take the system requirements and identify and label the software parts. When the parts are labeled, the baseline process will begin. Next CC will offer an organized method to make changes in the system. SA will record and report the history of the system. The APSE data base will be the primary mechanism for this. The toolset will assist in performing not only SA but also CC and CA. CA is done to verify and validate the system.

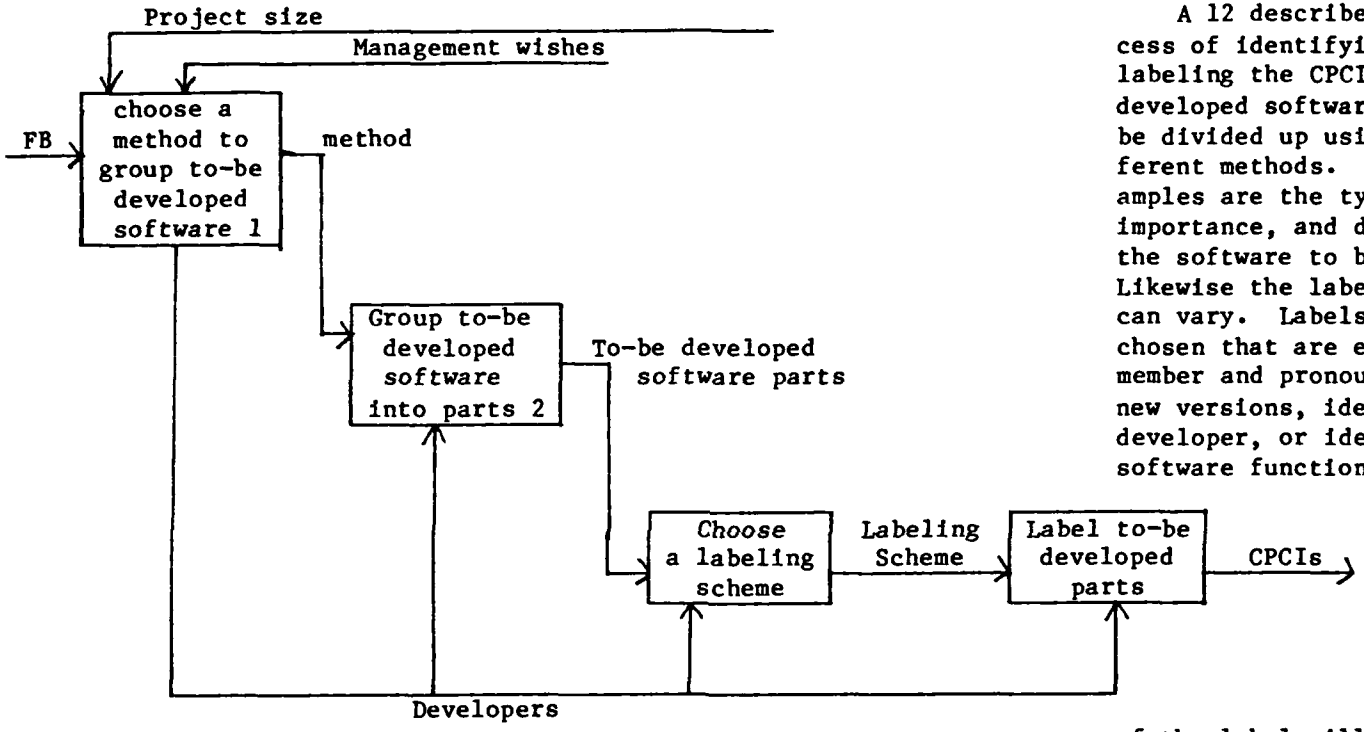


NODE A 1 TITLE - Perform Configuration Identification

A 1 describes CI. Writing the CI documents is a manual task done usually by software engineers. The writer's task can be made easier by such automated tools as word processors. An arrow head enclosed in parentheses indicate that the arrow is not shown in the parent diagram. In A 1 the proper names of the baselines are used. In A 0 the baselines were described as an entity.

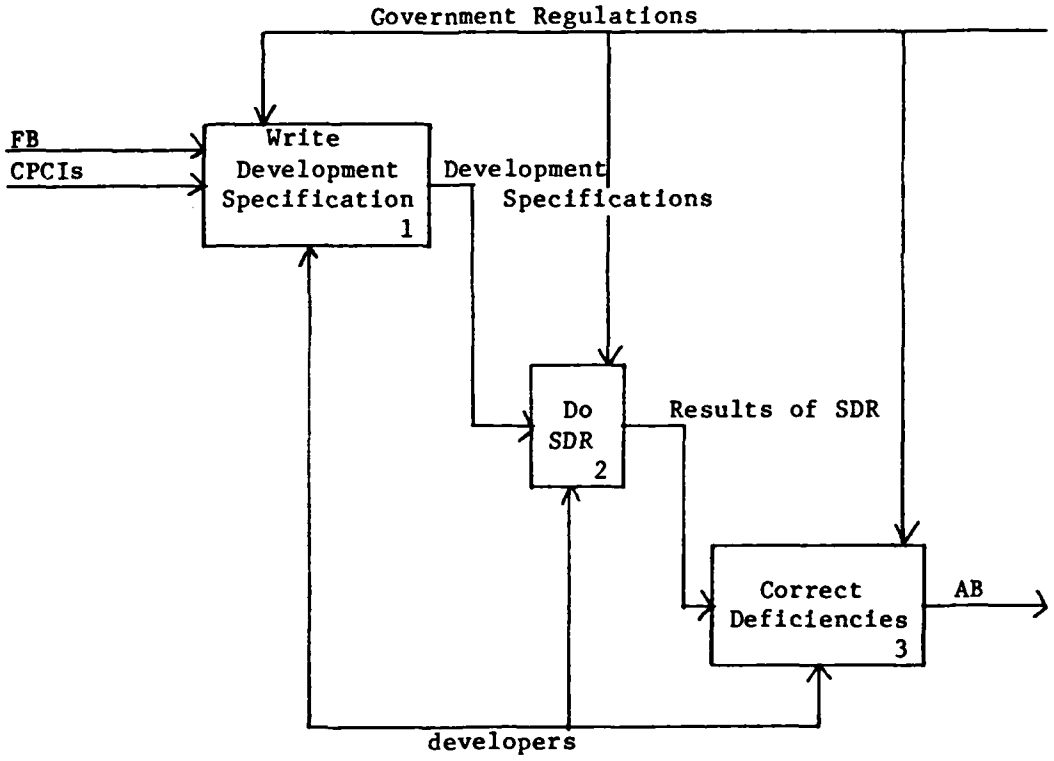


NODE A 11 TITLE - Prepare FB

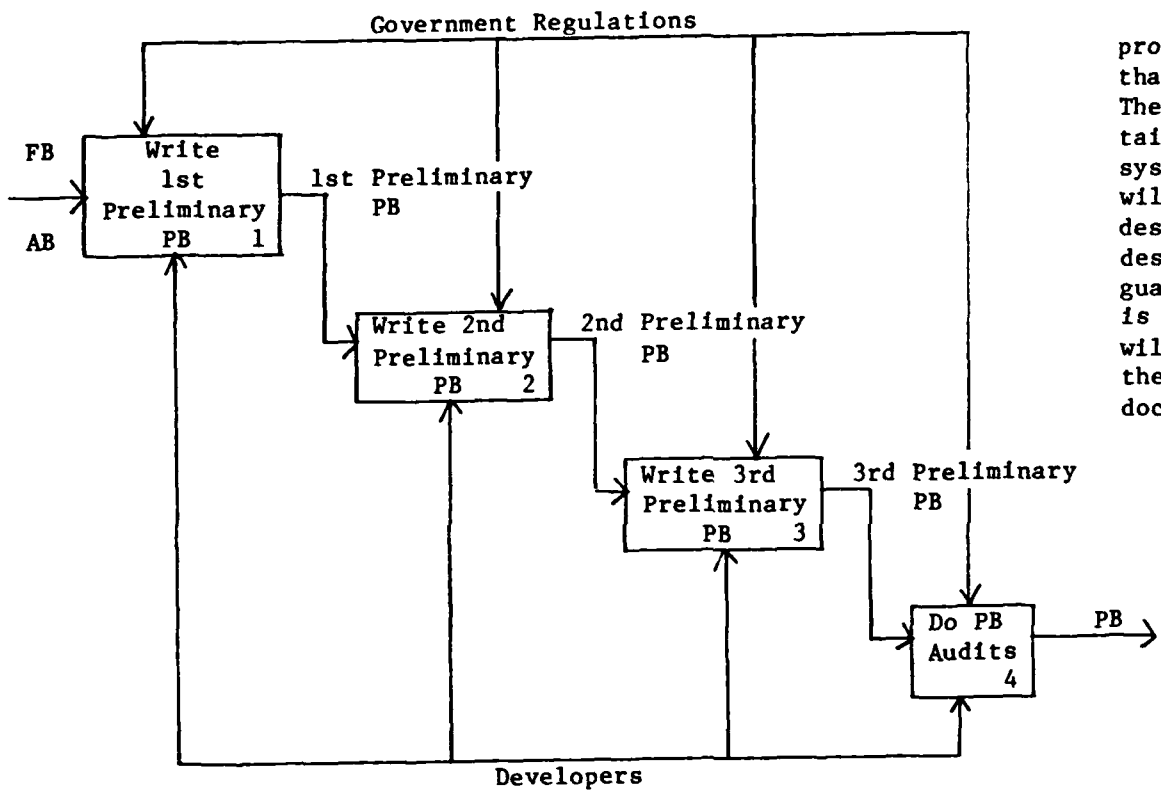


A 12 describes the process of identifying and labeling the CPCIs. A to-be-developed software system can be divided up using many different methods. Some examples are the type, size, importance, and difficulty of the software to be developed. Likewise the labeling schemes can vary. Labels can be chosen that are easy to remember and pronounce, reflect new versions, identify the developer, or identify the software function. The size of the label will not be a factor in the APSE.

NODE A 12 TITLE - Identify and Label CPCIs

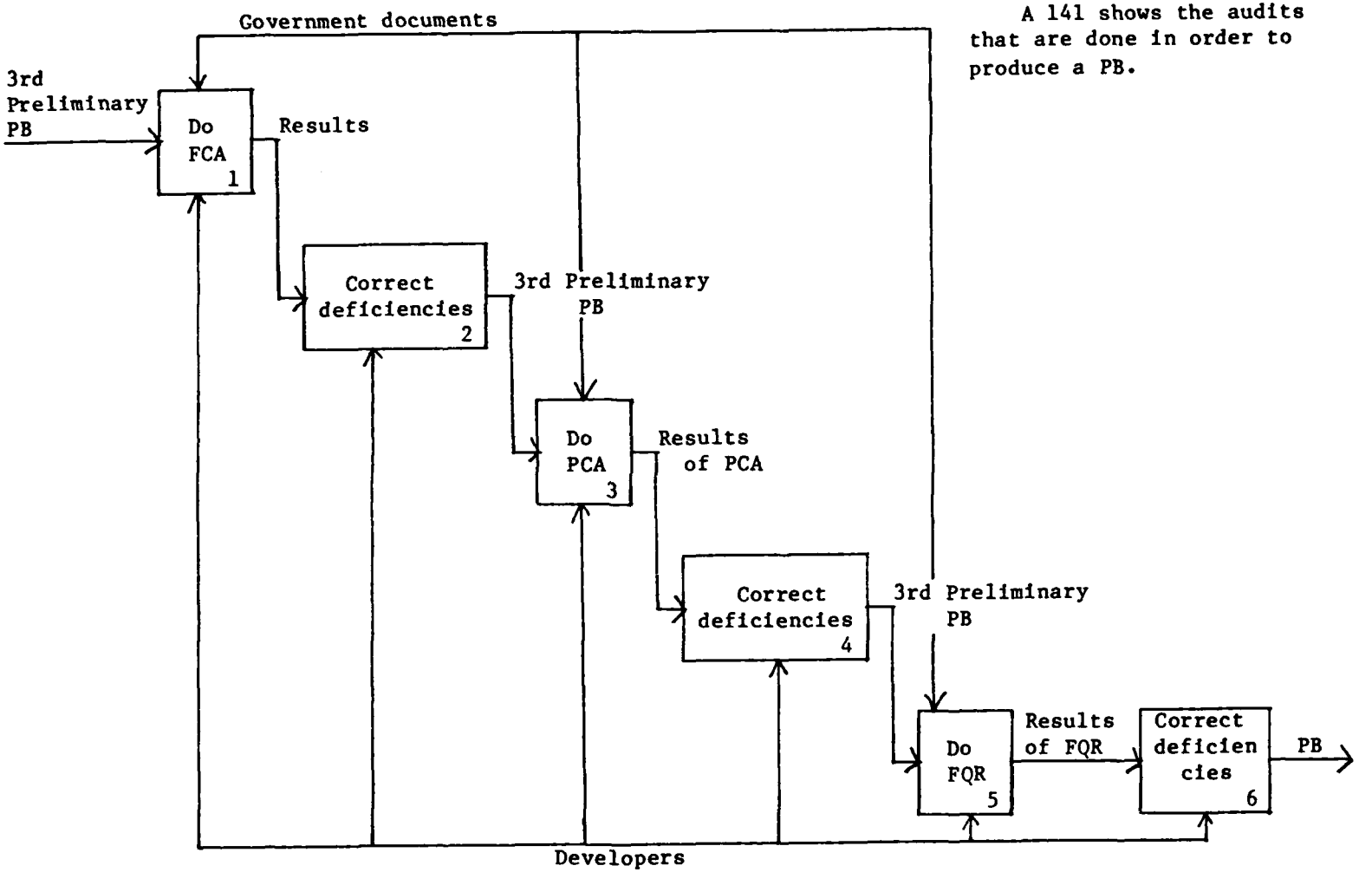


NODE A 13 TITLE - Prepare AB



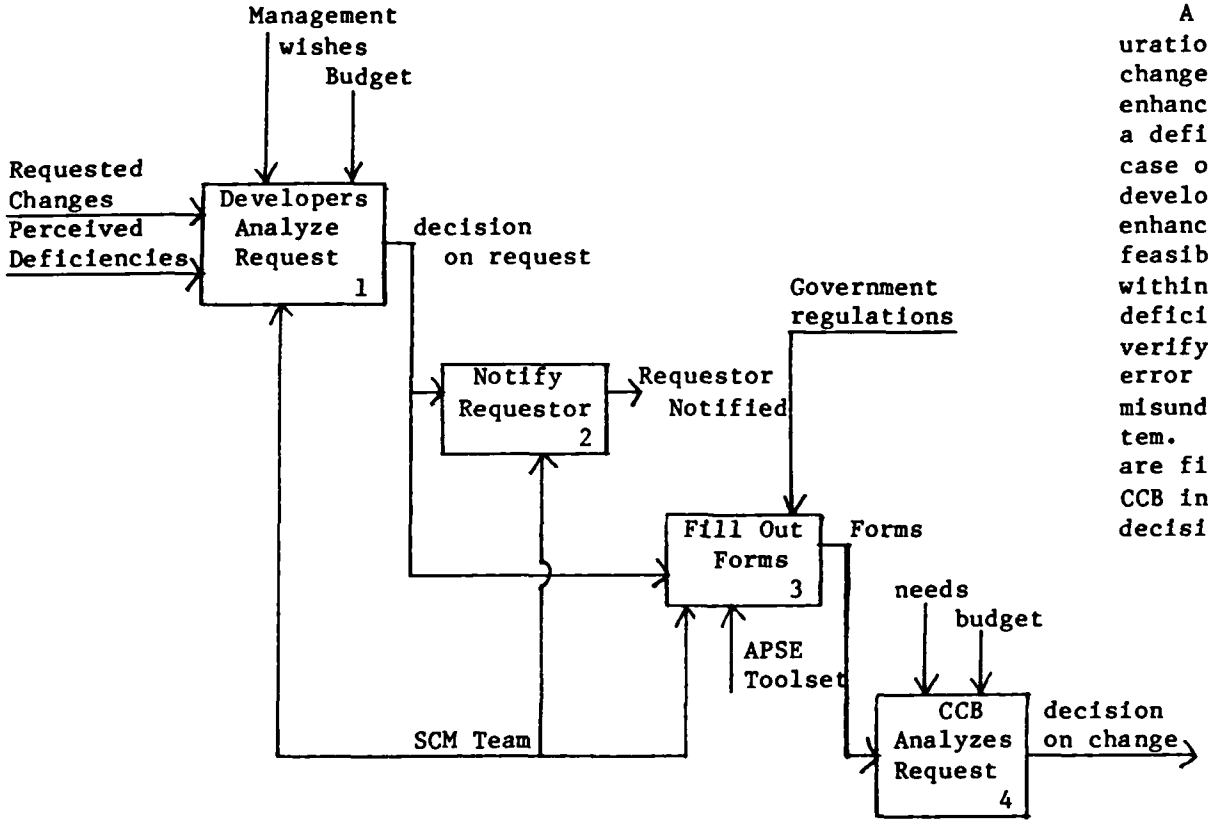
A 14 describes the PB process. It is more complex than the other baselines. The first version will contain a detailed design of the system. The second version will contain the technical descriptions and the system design done in a design language. The third document is the final version. It will include the code and the accompanying documentation.

NODE A 14 TITLE - Prepare PB



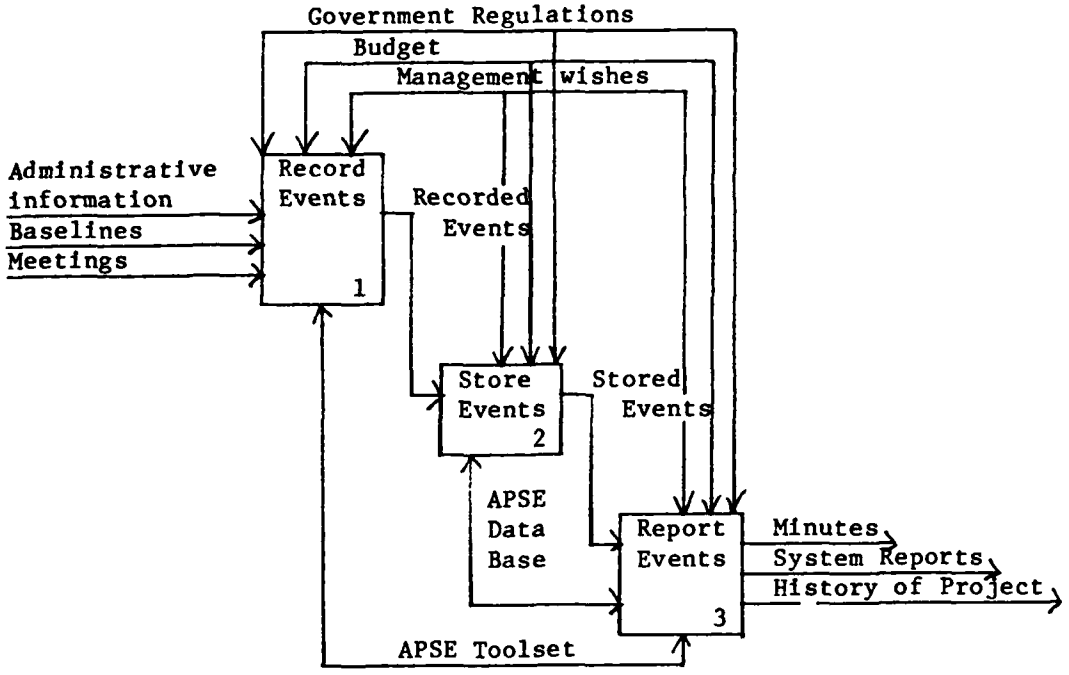
A 141 shows the audits that are done in order to produce a PB.

NODE A 141 TITLE - Do PB Audits



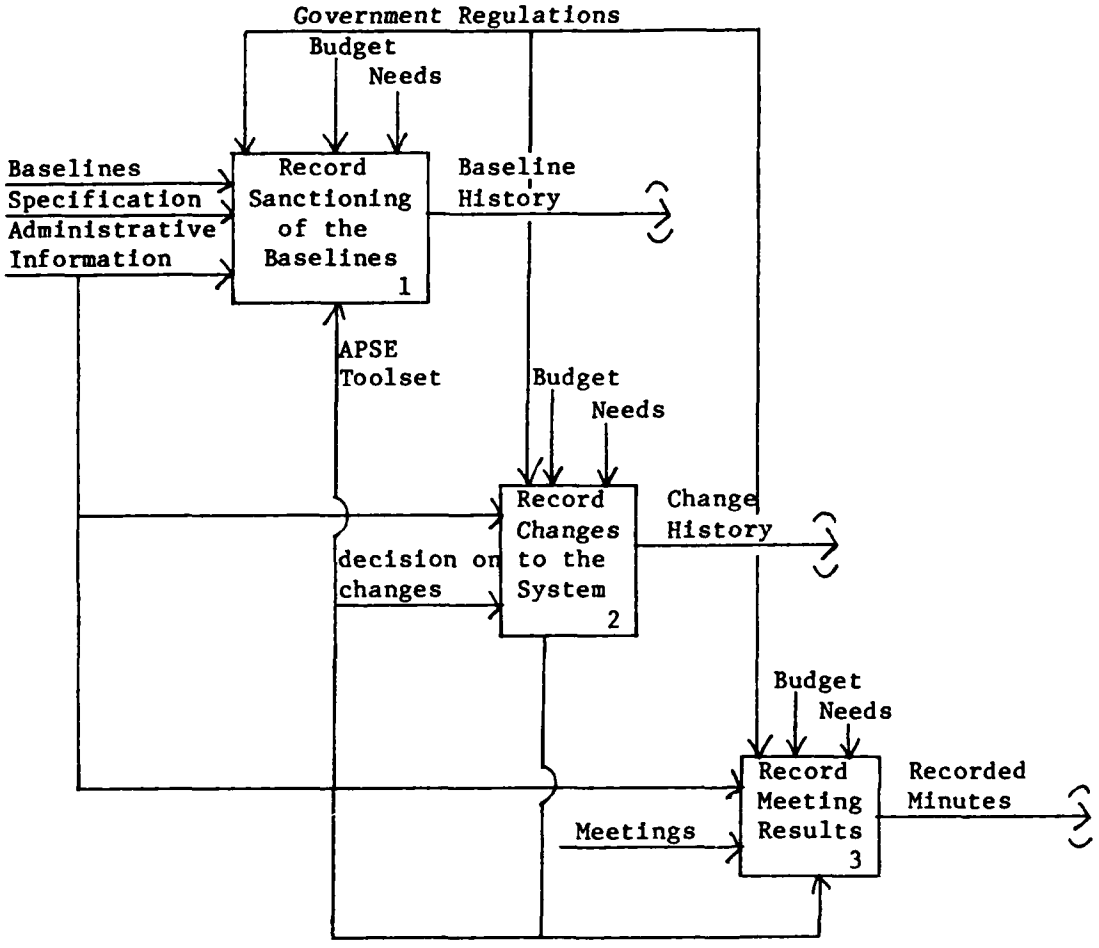
NODE A 2 TITLE - Perform Configuration Control

A 2 describes the configuration control process a change can be requested to enhance the system or correct a deficiency in it. In the case of an enhancement, the developers ensure that the enhancement is technically feasible and can be done within budget. A perceived deficiency is checked to verify that it really is an error and not just a user's misunderstanding of the system. A set of required forms are filled out to assist the CCB in making the final decision.



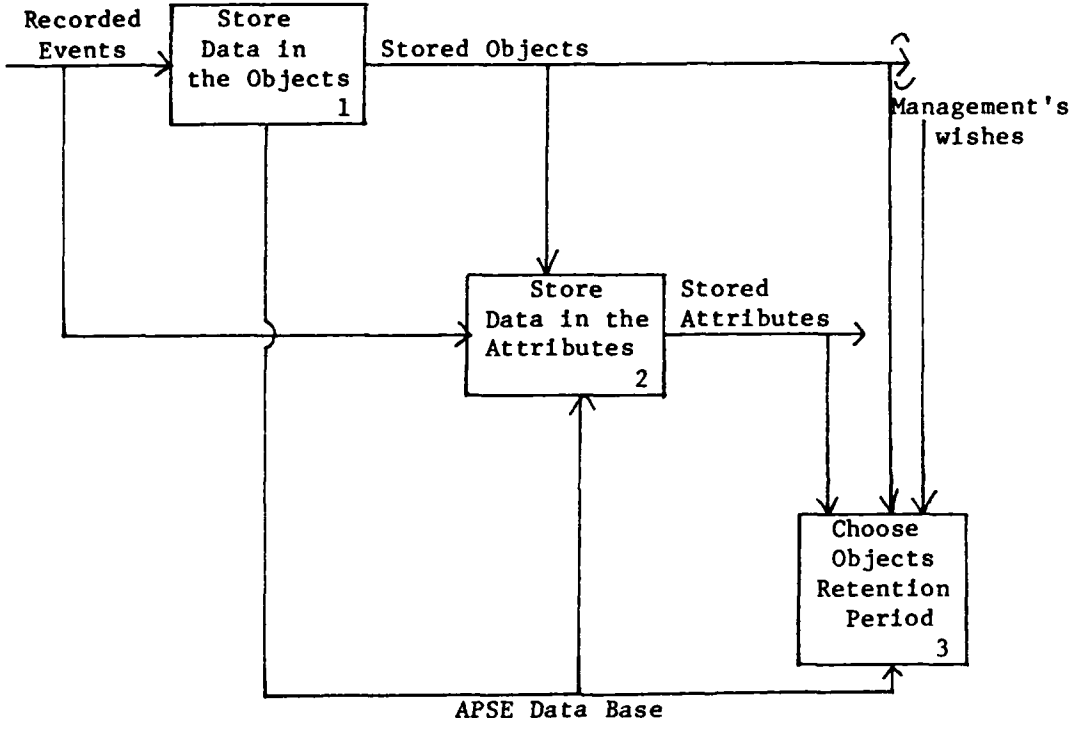
NODE A 3 TITLE - Perform Status Accounting

A 3 shows the process of status accounting. The history of the project will be stored in the APSE database and maintained and generated by the APSE toolset. Of all functions of SCM, SA will be the most automated.



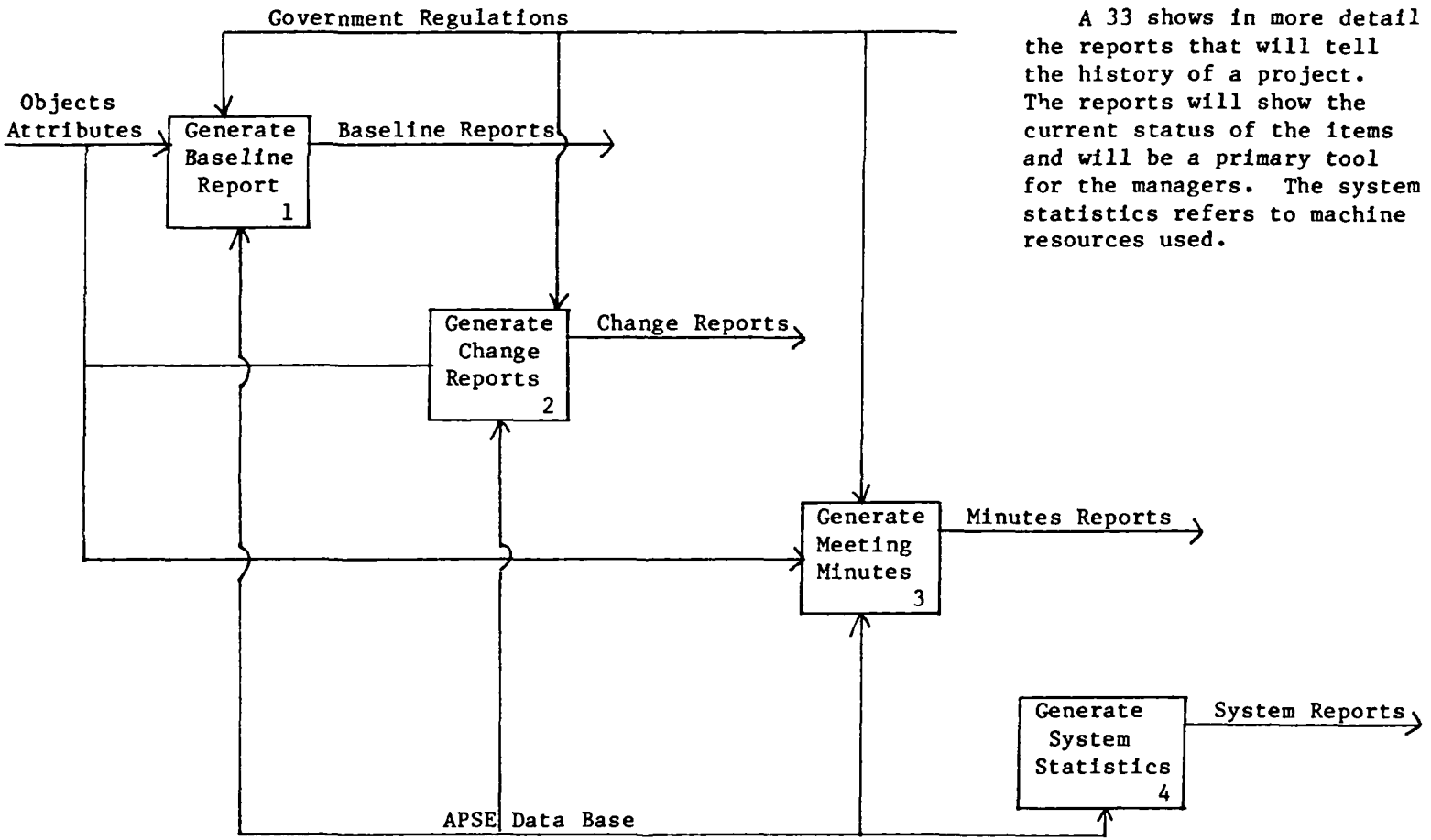
NODE A 31 TITLE - Record Events

A 31 breaks up the events to be recorded into three main categories. They are the baselines, changes, and meetings. These categories will be able to present the history of a project. The baseline data would include such information as who wrote the to-be-established baseline and when it was accepted as the baseline. The change data would include who requested the change, what the change involved, the forms filled out, who did the change, and when it was implemented. Meeting data would include the results of all the major meetings associated with the project. The extent of the information recorded depends on the budget, the complexity of the project, and the amount of control management wishes to exert.



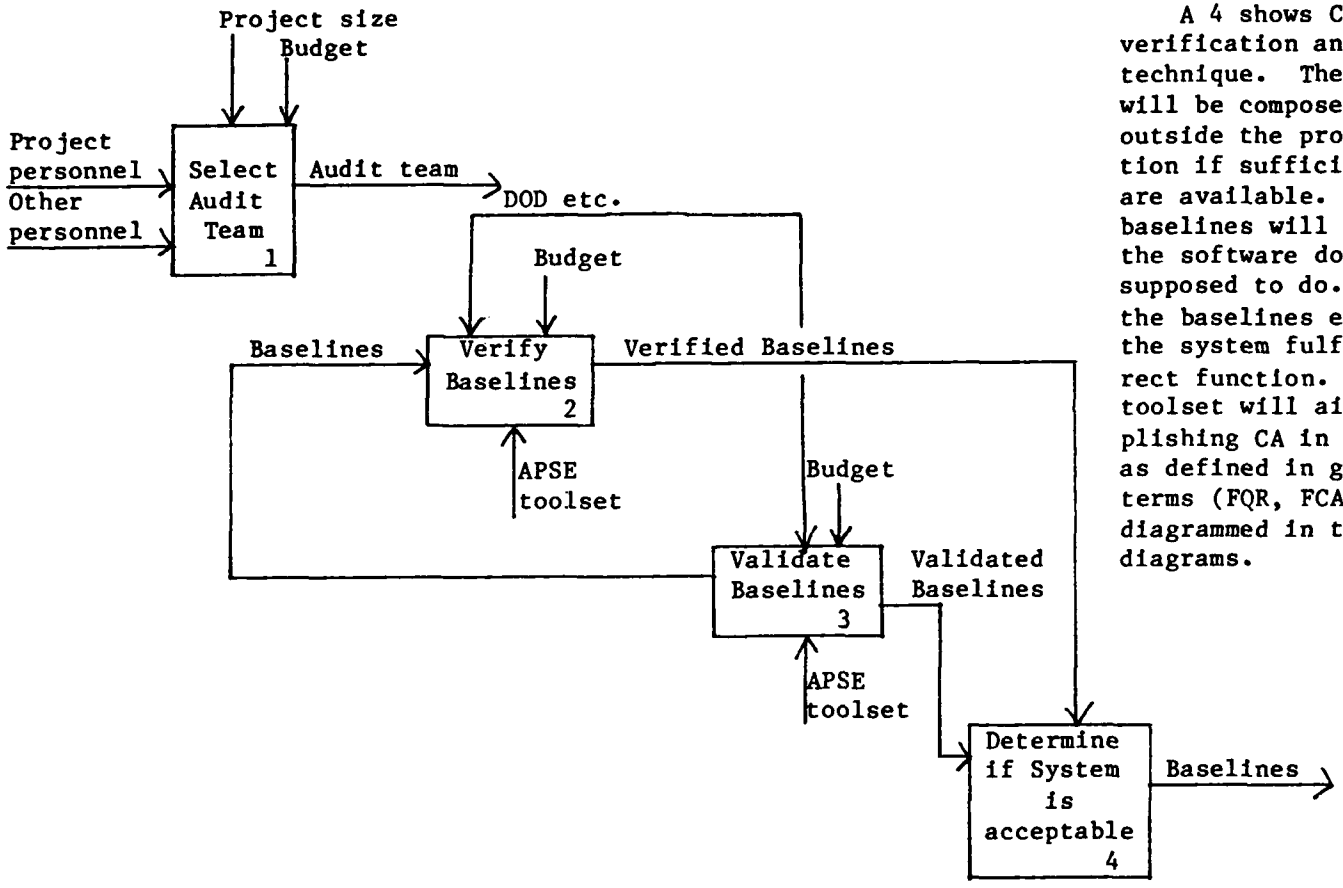
NODE A 32 TITLE - Store Events

A 32 describes storing the events. First, data is stored in the objects. This action includes saving and naming the object. The attributes associated with the object are then filled out. The attributes will contain administrative data concerning the object and who will be able to access the object. The manager then determines how long the object will remain in memory.



A 33 shows in more detail the reports that will tell the history of a project. The reports will show the current status of the items and will be a primary tool for the managers. The system statistics refers to machine resources used.

NODE A 33 TITLE - Report Events



A 4 shows CA done as a verification and validation technique. The audit team will be composed of people outside the project organization if sufficient resources are available. Verifying the baselines will ensure that the software does what it is supposed to do. Validating the baselines ensures that the system fulfills the correct function. The APSE toolset will aid in accomplishing CA in the APSE. CA as defined in government terms (FQR, FCA, PCA) is diagrammed in the A 1 (CI) diagrams.

NODE A 4 TITLE - Perform CA

VI Conclusions and Recommendations

Conclusions

From this research, the following conclusions were drawn:

1. SCM is a complex but important task in the software life cycle. Through it the cost of software can be reduced and the quality improved.

2. SCM is a relatively new discipline. Currently, very little guidance is offered on how to accomplish it. The APSE will correct this situation.

3. The APSE configuration manager will not completely automate the SCM task. But it will make it easier and less tedious to accomplish.

4. As with the other APSE tools the configuration manager will reside in the data base and share other APSE tools.

Recommendations

This paper can be used in several ways. First, it can be used as a tutorial for an overview of SCM and the APSE. The basic principles of each are covered. Second, it can be used to analyze other APSE configuration manager designs or systems. The requirements chapter can be used as a checklist to ensure that all of the requirements of Stoneman and the definition of SCM were met. Lastly, it can be used as a first step towards building and implementing an

APSE configuration manager. Both the background work and the preliminary design of what the tool must do is complete. The next step will be to design how to implement it. The emphasis in the implementation design should be in meeting the requirements of the MAPSE. Stoneman requires that the MAPSE configuration manager be able to allow access to the history attributes and ensure that the manager has control over the persistence of objects in the data base (Ref 10:38).

Bibliography

1. Aeronautical Systems Division. Airborne Systems Software Acquisition Engineering Guidebook for Application and Use of the Guidebooks. ASD-TR-80-5028. Wright-Patterson AFB OH, October 1980.

2. -----. Airborne Systems Software Acquisition Engineering Guidebook for Configuration Management. ASD-TR-79-5024. Wright-Patterson AFB OH, November 1978.

3. -----. Airborne Systems Software Acquisition Engineering Guidebook for Documentation Requirements. ASD-TR-79-5025. Wright-Patterson AFB OH, November 1978.

4. -----. Airborne Systems Software Acquisition Engineering Guidebook for Regulations, Specifications, and Standards. ASD-TR-78-6. Wright-Patterson AFB OH, November 1977.

5. -----. Airborne Systems Software Acquisition Engineering Guidebook for Reviews and Audits. ASD-TR-78-7. Wright-Patterson AFB OH, November 1978.

6. -----. Airborne Systems Software Acquisition Engineering Guidebook for Software Development Planning and Control. ASD-TR-80-5022. Wright-Patterson AFB OH, February 1980.

7. -----. Airborne Systems Software Acquisition Engineering Guidebook for Verification, Validation, and Certification. ASD-TR-79-5028. Wright-Patterson AFB OH, September 1978.

8. Bersoff, Edward H., Vilas D. Henderson and Stanley G. Siegel. Software Configuration Management An Investment in Product Integrity. Englewood Cliffs NJ: Prentice-Hall Inc., 1980.

9 Bryan, William, Christopher Chadbourne and Stan Siegel, eds. "Tutorial: Software Configuration Management," Initially presented at the IEEE Computer Society's Fourth International Computer Software & Applications Conference (COMPSAC 80). IEEE Catalog Number EHO 169-3. Falls Church VA: IEEE Computer Society, 27-31 October 1980.

10. Buxton, J. and V. H. Stenning, eds. "Stoneman," Requirements for Ada Programming Support Environment. U.S. Department of Defense, February 1980.

11. Carlson, William E. "Software Research in the Department of Defense," Proceedings of the 2nd International Conference on Software Engineering. IEEE Catalog Number 76ch1125-4c. 379-383. October 1976.

12. Clema, Joe K. and Larry Levsen, "Management and Control of Large-Scale Software Systems," Weapon System Software Acquisition, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH. 5-14 (1979).

13. Gunther, Richard C. Management Methodology for Software Product Engineering. New York: John Wiley & Sons Inc., 1978.

14. Knight, B. M. "Software Quality and Productivity," Weapon Systems Software Acquisition. Wright-Patterson AFB OH: Air Force Institute of Technology, 117-128.

15. Pyle, I. C. The Ada Programming Language. London: Prentice-Hall International Inc., 1981.

16. Rome Air Development Center (Intermetrics + Mass Comp Assoc). Ada Integrated Environment Design Rationale: Technical Report (Interim). 13 March 1981.

17. Rubey, Raymond J. SofTech, Inc. Course EE 5.45, "Software Acquisition," School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH. Lecture materials. 1982.

18. Stoner, Bill. Logicon. Course EE 5.45, "Software Acquisition," School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH. Lecture materials. 1982.

19. U.S. Department of the Air Force. Configuration Management. AFR 65-3. Washington: Government Printing Office, 1 July 1974.

20. U.S. Department of Defense. Configuration Management. DOD Directive 5019.19. Washington: Government Printing Office, 1 May 1979.

21. U.S. Department of Defense. Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs. Military Standard 483. Washington: Government Printing Office, 31 December 1970.

22. U.S. Department of Defense. Reference Manual for the Ada Programming Language Proposed Standard Document. Washington: Government Printing Office, July 1980.

CHANGE REQUEST

1. System name: _____ 2. Control no.: _____

3. Application level:

System ☐ Hardware ☐ Software ☐ Document ☐ Other ☐

4. A. Originating organization	5. CI affected (highest level)	6. Documents affected A. _____ D. _____ B. _____ E. _____ C. _____ F. _____
B. Initiator		
C. Telephone #	7. Priority: A. Routine ☐ B. Urgent ☐ C. Emergency ☐	8. Other systems/ software/equipment affected Yes ☐ No ☐ If yes, explain in block 9.C
D. Date		

9. Narrative

A. Description of change

B. Need for change

C. Estimated effects on other systems/software/equipment

D. Alternatives

To be completed by cognizant CM manager

10. Date received	12. Disposition
11. ECP requested Yes ☐ No ☐	13. Signature 14. Date

Appendix A - CR Ref (8:201)

SOFTWARE INCIDENT REPORT

1. System name: _____ 2. Control no.: _____

Part I

3. User activity	4. Incident occurred	5. Software identification (if known)
A. Organization	A. Date	A. Name/acronym
B. Initiator	B. Time	B. Module/subroutine
C. Telephone #	6. Related SIR Superseded ☐ Modified ☐	C. Software function
D. Date		7. Urgency high ☐ med ☐ low ☐ n/a ☐

8. Problem category	9. Affected documents	10. Related documents
Document ☐ Executable code ☐	A._____ D._____ B._____ E._____ C._____ F._____	A._____ B._____

11. Media ID no.___	12. Executable code address	13. Duplicate media tested	14. Problem duplicated
Tape ☐ Cards ☐ Disk ☐		yes ☐ no ☐	yes / no / n/a During run After restart After reload

15. Test spec reference	16. Dump data information

17. Description of incident, effects, and recommendations

Part II

18. Developing activity	19. Responsible organization/name	20. Telephone #

21. Analysis/corrective action

22. Disposition:

☐ NAR ☐ DTF ☐ SCN # ___ ☐ CR # ___

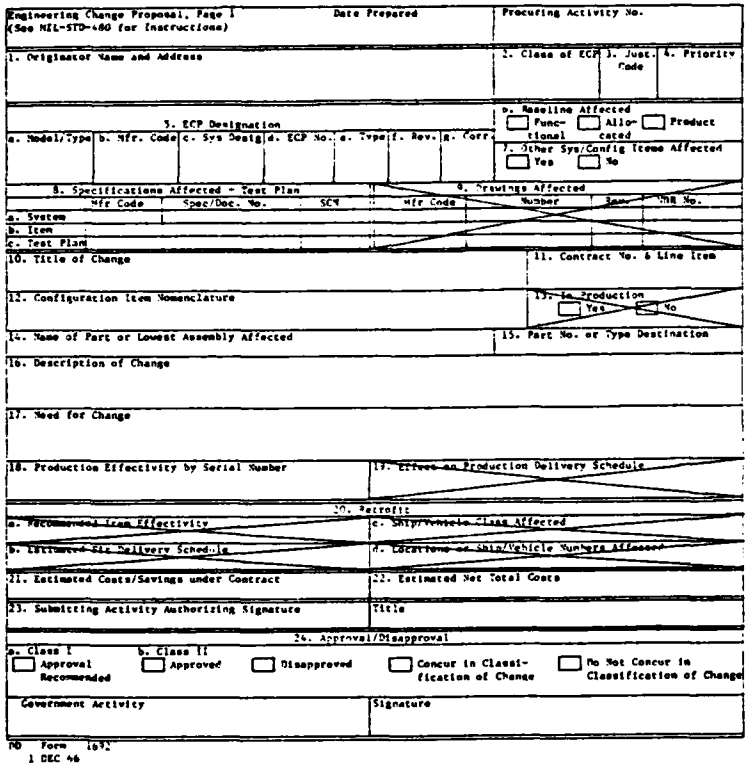
Documents affected:

A. _____ D. _____
B. _____ E. _____
C. _____ F. _____

23. Comments to initiator:

Appendix A - SIR Ref (8:202)

ECP FORM FOR A COMPUTER PROGRAM



Engineering Change Proposal, Page 1 (See MIL-STD-480 for Instructions) | Date Prepared | Procuring Activity No.

1. Originator Name and Address | 2. Class of ECP | 3. Just. Code | 4. Priority

5. ECP Designation: a. Model/Type | b. Mfr. Code | c. Sys Desig | d. ECP No. | e. Type | f. Rev. | g. Corr.

6. Baseline Affected: ☐ Functional ☐ Allocated ☐ Product

7. Other Sys/Config Items Affected: ☐ Yes ☐ No

8. Specifications Affected – Test Plan: Mfr Code | Spec/Doc. No. | SCN

9. Drawings Affected: Mfr Code | Number | Rev. | Dwg No.

a. System

b. Item

c. Test Plan

10. Title of Change | 11. Contract No. & Line Item

12. Configuration Item Nomenclature | 13. In Production: ☐ Yes ☐ No

14. Name of Part or Lowest Assembly Affected | 15. Part No. or Type Designation

16. Description of Change

17. Need for Change

18. Production Effectivity by Serial Number | 19. Effect on Production Delivery Schedule

20. Retrofit

a. Recommended Item Effectivity | c. Ship/Vehicle Class Affected

b. Estimated Kit Delivery Schedule | d. Locations of Ship/Vehicle Numbers Affected

21. Estimated Costs/Savings under Contract | 22. Estimated Net Total Costs

23. Submitting Activity Authorizing Signature | Title

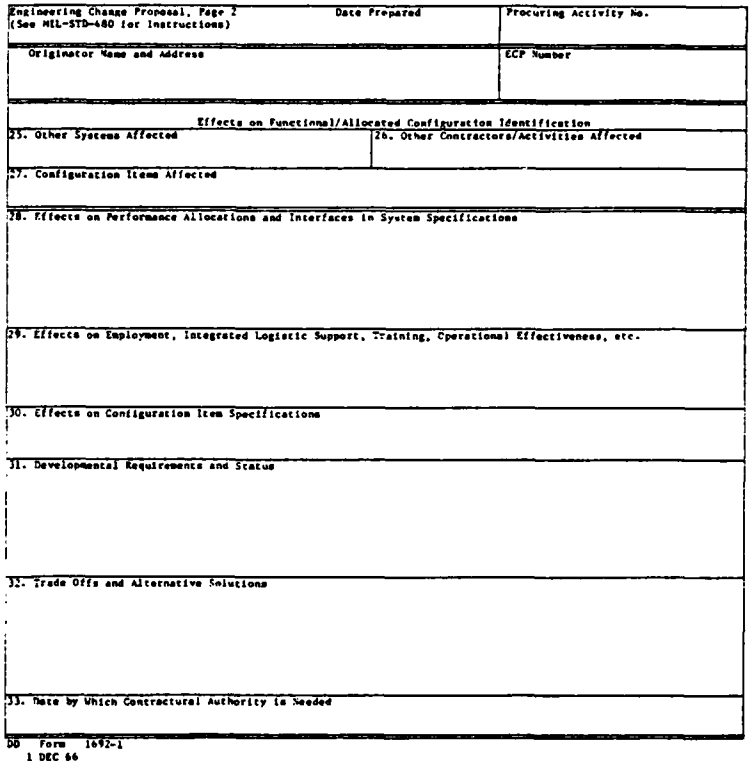
24. Approval/Disapproval

a. Class I: ☐ Approval Recommended

b. Class II: ☐ Approved ☐ Disapproved ☐ Concur in Classification of Change ☐ Do Not Concur in Classification of Change

Government Activity | Signature

DD Form 1692
1 DEC 66



Engineering Change Proposal, Page 2 (See MIL-STD-480 for Instructions) | Date Prepared | Procuring Activity No.

Originator Name and Address | ECP Number

Effects on Functional/Allocated Configuration Identification

25. Other Systems Affected | 26. Other Contractors/Activities Affected

27. Configuration Items Affected

28. Effects on Performance Allocations and Interfaces in System Specifications

29. Effects on Employment, Integrated Logistic Support, Training, Operational Effectiveness, etc.

30. Effects on Configuration Item Specifications

31. Developmental Requirements and Status

32. Trade Offs and Alternative Solutions

33. Date by Which Contractural Authority is Needed

DD Form 1692-1
1 DEC 66

SOFTWARE CONFIGURATION MANAGEMENT PLAN

Data Item Description	2. Identification No(s) Agency	Number
1. Title Software Configuration Management Plan	Navy	DI-E-2175
3. Description/Purpose 3.1 The Software Configuration Management Plan (SCMP) describes the contractor's internal computer software configuration management organization; the responsibility of the members; the relationship among the several offices/divisions; the policies and procedures for identifying the documentation of the functional and physical characteristics of configuration items required by the contract; procedures for controlling changes to configuration items during development; (continued on page 2)	4. Approval Date 29 November 1978	
	5. Office of Primary Responsibility NM(MAT-09Y)	
	6. DDC Required	
	8. Approval Limitation	
7. Application/Interrelationship 7.1 The Software Configuration Management Plan provides the contractor the means to consolidate all policies, procedures, organizational descriptions, resources and schedules relating to software configuration management in one document. The SCMP provides the procuring activity with detailed knowledge of the contractor's configuration management. Through the SCMP the procuring activity can monitor the contractor's application of configuration management principles in conformance with standards invoked in the contract. 7.2 This Data Item Description supersedes UDI-E-22191.	9. References (Mandatory as cited in block 10) MIL-STD-1679 (Navy)	
	MCSL Number (s)	

10. Preparation Instructions.

10.1 Unless otherwise stated in the solicitation, the effective date of the document(s) cited in this block shall be that listed in the issue of the DoD Index of Specifications and Standards (DoDISS) and the supplements thereto specified in the solicitation and will form a part of this Data Item Description to the extent defined within.

10.2 Content and Format Instructions. The Software Configuration Management Plan shall be in accordance with the following content and format instructions:

SECTION 1 - Introduction.

1.1 Purpose and Scope. This paragraph shall state the purpose, scope, and general applicability of the SCMP.

1.2 Definitions. This paragraph shall reference applicable directives or glossaries containing definitions of terminology used in the SCMP and shall further define any terms used which are not contained herein.

1.3 Configuration Management Summary. This paragraph shall provide a concise summary of the approach used to accomplish configuration management. Describe the plan's major features and objectives.

DD FORM 1664 S/N 0102-LF-039-4000 Page 1 of 3 pages
1 JUN 68

Appendix B - DID for CM Plan Ref (9)

DI-E-2175
3. DESCRIPTION/PURPOSE (continued)

procedures for recording and reporting change processing implementation status; and the external relationships required to maintain total system compatibility.

---

10. PREPARATION INSTRUCTIONS (continued)

SECTION 2 - Applicable Documents.

This section shall list those specifications, standards, manuals, and other documents applicable to the configuration management effort. Each document shall be completely identified by title, document number, issuing authority, and date of issue.

SECTION 3 - Software Configuration Management (CM) Organization

This section shall identify the contractor's organization for CM. It shall show the:

a. Relationships among the contractor's project organization, functional organizations, and facility management.

b. Responsibilities and authority for CM of all participating groups and organizations.

c. Identification of contractor CM organization including configuration control boards, both internal and external.

d. Policies and directives relating to CM.

e. Relationships among the contractor's software CM organization, the contractor's hardware CM organization, and the project's hardware CM organization when the software is only one element of the weapon system being developed.

SECTION 4 - Software Configuration Identification.

This section shall present the contractor's implementation plans for:

a. Selecting and identifying configuration items, as required by the contract, and additional items considered necessary by the contractor to ensure proper configuration identification.

b. Developing, numbering, changing, and maintaining specifications and their relationship with specification trees.

c. Establishing internal baselines as appropriate.

d. Preparing and processing of design specifications during development and their identification and relationship to higher level specifications or documentation.

e. Establishing the development support library.

f. Assignment of nomenclature and serial numbers.

SECTION 5 - Software Configuration Control.

This section shall describe the contractor's organization and procedures for:

a. Configuration control, including depth of control, interfaces, and subcontractor/vendor control.

DI-E-2175
10. PREPARATION INSTRUCTIONS (continued)

b. Preparation, processing and submittal to the contractor's internal configuration control board of Software Change Proposals (SCP), Software Enhancement Proposals (SEP) and Engineering Change Proposals (ECP).

c. Preparation, processing and submittal to procuring agency or the procuring agency's representative configuration control board of SCPs, SEPs, and ECPs.

d. Promulgation and implementation of specification change notices.

e. Preparation and processing of Software Trouble Reports (STR).

f. Ensuring that the implementation of approved changes is reflected in all facets of the affected baselines, program descriptive documentation, and program materials (e.g., design, test, and user narrative).

g. The contractor's software configuration control board.

SECTION 6 - Software Configuration Authentication.

This section shall describe the contractor's procedures for:

a. Reconciling deliverable software to its approved documentation.

b. Assuring that the software, descriptive documentation, and program materials are properly identified.

c. Assuring the incorporation of approved changes.

d. Reconciling the configuration status accounting reports and the status of the software, descriptive documentation, and program materials with the approved baseline(s) and its approved changes.

SECTION 7 - Software Configuration Status Accounting.

This section shall present the contractor's procedures for collecting, recording, processing, and maintaining data necessary for producing configuration status accounting reports. It shall include:

a. Formats and data elements for software CM status accounting records and reports.

b. Content and format of periodic summary reports to reflect status of SCPs, SEPs, and STRs as appropriate.

SECTION 8 - Interface Management.

This section shall describe the contractor's plan for coordinating efforts involved in design and data management to ensure compatibility through interfaces with associated contractors.

SECTION 9 - Subcontractors & Vendors.

This section shall present the contractor's system for control over subcontractors and vendors. In particular, it shall explain the capability of subcontractors/vendors to support the requirements of Configuration Management. It shall enumerate the requirements and provisions for review and approval of all changes submitted by subcontractors/vendors to comply with established procedures.

Appendix B - DID for CM Plan Ref (9)
(continued)

Vita

Susan Mary Schultz was born on 8 June 1953 in Royal Oak, Michigan. She graduated from Royal Oak Dondero High School in 1971. She received a Bachelor of Arts degree in Math in 1976 from Wayne State University in Detroit, Michigan. She entered the Air Force on active duty in 1978 and, in November of that year, received her commission from Officer Training School. Until entering the School of Engineering, Air Force Institute of Technology in June 1981, she served as a Computer Systems Programs Officer at Headquarters Tactical Air Command, Joint Studies Group, Nellis Air Force Base, Nevada.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ~~GCS~~/MA/82D-12	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19	
12. PERSONAL AUTHOR(S) Susan M. Schultz, B.A., Capt, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1984 June 15. PAGE COUNT 70
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Ada, Configuration Managment, Ada Programming Support Enviroment (APSE).

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

Title: Preliminary Design of The Ada Programming Support Enviroment Configuration Manager

Thesis Chairman: Patricia Lawlis, Captain, USAF

Approved for public release: IAW AFR 190-17.

LYNN E. WELAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Patricia Lawlis, Captain, USAF	22b. TELEPHONE NUMBER (Include Area Code) 513-255-3636 22c. OFFICE SYMBOL AFIT/ENC

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

Today the development and maintenance of software are becoming prohibitively costly. With the goal of reducing the cost of producing a software system without sacrificing the quality of it, the Department of Defense (DOD) is sponsoring the development of the Ada Programming Support Environment (APSE). This paper explains the APSE. It also explains the requirements and gives a preliminary design of one of the major tools of the APSE, the configuration manager. The preliminary design of this tool is presented using Structured Analysis and Design (SADT) diagrams. The preliminary design includes only a functional description of the configuration manager. How to implement it is left for further research.

Prior to presenting the preliminary design of the configuration manager, a description of how SCM is currently practiced is given. SCM is divided into four functions. They are configuration identification (CI), configuration control (CC), status accounting (SA), and configuration auditing (CA). SA is the only function that can be completely automated. Therefore, the preliminary design emphasizes the SA function.

END

FILMED

4-85

DTIC