ESA PSS-05-09 Issue 1 Revision 1
March 1995

Guide to software configuration management

Prepared by:
ESA Board for Software
Standardisation and Control
(BSSC)

european space agency / agence spatiale européenne
8-10, rue Mario-Nikis, 75738 PARIS CEDEX, France

DOCUMENT STATUS SHEET

DOCUMENT STATUS SHEET			
1. DOCUMENT TITLE: **ESA PSS-05-09 Guide to Software Configuration Management**			
2. ISSUE	3. REVISION	4. DATE	5. REASON FOR CHANGE
1	0	1992	First issue
1	1	1995	Minor updates for publication

Issue 1 Revision 1 approved, May 1995
Board for Software Standardisation and Control
M. Jones and U. Mortensen, co-chairmen

Issue 1 approved, 9th December 1992
Telematics Supervisory Board

Issue 1 approved by:
The Inspector General, ESA

Published by ESA Publications Division,
ESTEC, Noordwijk, The Netherlands.
Printed in the Netherlands.
ESA Price code: E1
ISSN 0379-4059

TABLE OF CONTENTS

PREFACE

This document is one of a series of guides to software engineering produced by the Board for Software Standardisation and Control (BSSC), of the European Space Agency. The guides contain advisory material for software developers conforming to ESA's Software Engineering Standards, ESA PSS-05-0. They have been compiled from discussions with software engineers, research of the software engineering literature, and experience gained from the application of the Software Engineering Standards in projects.

Levels one and two of the document tree at the time of writing are shown in Figure 1. This guide, identified by the shaded box, provides guidance about implementing the mandatory requirements for software configuration management described in the top level document ESA PSS-05-0.

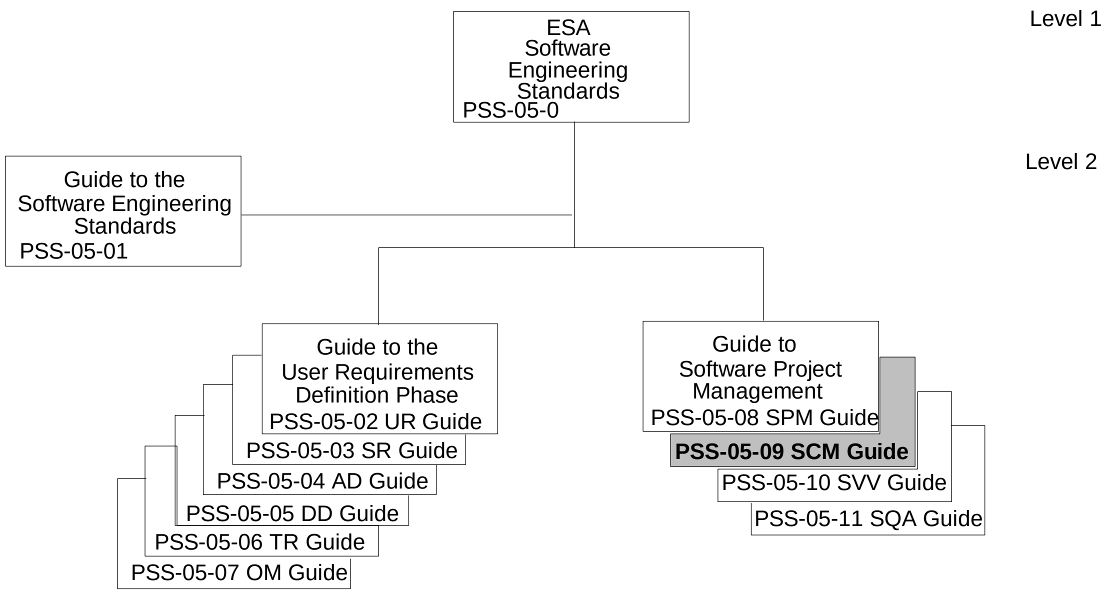

Figure 1: ESA PSS-05-0 document tree

The Guide to the Software Engineering Standards, ESA PSS-05-01, contains further information about the document tree. The interested reader should consult this guide for current information about the ESA PSS-05-0 standards and guides.

The following past and present BSSC members have contributed to the production of this guide: Carlo Mazza (chairman), Gianfranco Alvisi, Michael Jones, Bryan Melton, Daniel de Pablo and Adriaan Scheffer.

The BSSC wishes to thank Jon Fairclough for his assistance in the development of the Standards and Guides, and to all those software engineers in ESA and Industry who have made contributions.

Requests for clarifications, change proposals or any other comment concerning this guide should be addressed to:

BSSC/ESOC Secretariat
Attention of Mr C Mazza
ESOC
Robert Bosch Strasse 5
D-64293 Darmstadt
Germany

BSSC/ESTEC Secretariat
Attention of Mr B Melton
ESTEC
Postbus 299
NL-2200 AG Noordwijk
The Netherlands

CHAPTER 1 INTRODUCTION

1.1 PURPOSE

ESA PSS-05-0 describes the software engineering standards to be applied for all deliverable software implemented for the European Space Agency (ESA), either in house or by industry [Ref. 1].

ESA PSS-05-0 requires that the configuration of the code and documentation be managed throughout the software development life cycle. This activity is called Software Configuration Management' (SCM). Each project must define its Software Configuration Management activities in a Software Configuration Management Plan (SCMP).

This guide defines and explains what software configuration management is, provides guidelines on how to do it, and defines in detail what a Software Configuration Management Plan should contain.

This guide should be read by everyone concerned with developing, installing and changing software, i.e. software project managers, software librarians and software engineers. Some sections describing procedures for handling software problems and changing documents will be of interest to users.

1.2 OVERVIEW

Chapter 2 discusses the principles of software configuration management, expanding upon ESA PSS-05-0. Chapter 3 discusses tools for software configuration management. Chapter 4 describes how to write the SCMP.

All the mandatory practices in ESA PSS-05-0 relevant to software configuration management are repeated in this document. The identifier of the practice is added in parentheses to mark a repetition. This document contains no new mandatory practices.

This page is intentionally left blank.

CHAPTER 2
SOFTWARE CONFIGURATION MANAGEMENT

2.1 INTRODUCTION

The purpose of software configuration management is to plan, organise, control and coordinate the identification, storage and change of software through development, integration and transfer (SCM02). Every project must establish a software configuration management system. All software items, for example documentation, source code, executable code, files, tools, test software and data, must be subjected to SCM (SCM01). Software configuration management must ensure that:

- software components can be identified;
- software is built from a consistent set of components;
- software components are available and accessible;
- software components never get lost (e.g. after media failure or operator error);
- every change to the software is approved and documented;
- changes do not get lost (e.g. through simultaneous updates);
- it is always possible to go back to a previous version;
- a history of changes is kept, so that is always possible to discover who did what and when.

Project management is responsible for organising software configuration management activities, defining software configuration management roles (e.g. software librarian), and allocating staff to those roles. In the TR and OM phases, responsibility for software configuration management lies with the Software Review Board (SRB).

Each group has its own characteristic requirements for software configuration management [Ref. 5]. Project management requires accurate identification of all items, and their status, to control and monitor progress. Development personnel need to share items safely and efficiently. Quality assurance personnel need to be able to trace the derivation of each item and establish the completeness and correctness of each configuration. The configuration management system provides visibility of the product to everyone.

In large developments, spread across multiple hardware platforms, configuration management procedures may differ in physical details. However, a common set of configuration management procedures must be used (SCM03).

No matter how large the project may be, software configuration management has a critical effect on quality. Good software configuration management is essential for efficient development and maintenance, and for ensuring that the integrity of the software is never compromised. Bad software configuration management can paralyse a project. Sensible change requests may fail to be approved because of fears that they cannot be implemented correctly and will degrade the system.

This chapter summarises the principles of software configuration management described in ESA PSS-05-0 and then discusses the application of these principles first to documents and then to code.

2.2 PRINCIPLES OF SOFTWARE CONFIGURATION MANAGEMENT

The subject matter of all software configuration management activities is the 'configuration item' (CI). A CI is an aggregation of hardware, software or both that is designated as a unit for configuration management, and treated as a single entity in the configuration management process [Ref. 2]. A CI can be an aggregation of other CIs, organised in a hierarchy. Any member of this hierarchy can exist in several versions, each being a separate CI. For code, the CI hierarchy will normally be based on the design hierarchy, but may be modified for more efficient configuration management. Figure 2.2.A shows an example CI hierarchy.

Examples of configuration items are:

- software component, such as a version of a source module, object module, executable program or data file;
- support software item, such as a version of a compiler or a linker;
- baseline, such as a software system under development;
- release, such as a software system in operational use;
- document, such as an ADD.

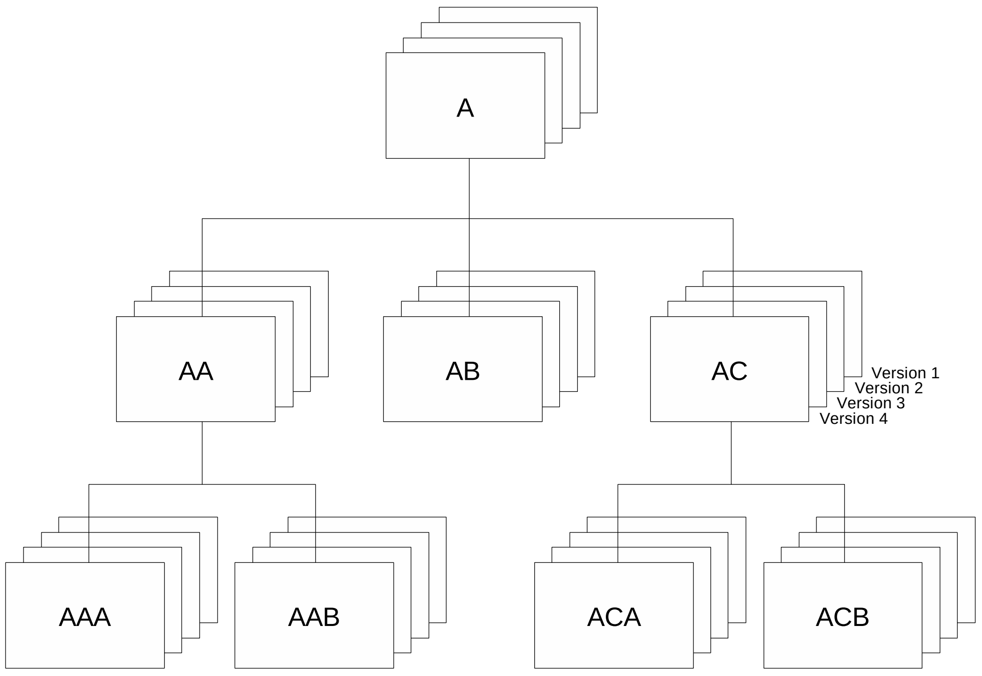

Figure 2.2.A: Example Configuration Item hierarchy

Software configuration management is formally defined in ANSI/IEEE Std 610.12-1990 [Ref. 2] as the discipline of applying technical and administrative direction and administration to:

- identify and document the functional and physical characteristics of a configuration item;
- control changes to those characteristics;
- record and report change processing and implementation status;
- verify compliance with specified requirements.

All configuration items, from modules to entire software releases, must be defined as early as possible and systematically labelled when they are created. Software development can only proceed smoothly if the development team are able to work on correct and consistent configuration items. Responsibilities for the creation and modification of configuration items must be allocated to individuals and respected by others. Although this allocation is defined in the Work Breakdown Structure (WBS) of the Software Project Management Plan (SPMP), the Software Configuration Management Plan (SCMP) defines the procedures that coordinate the efforts of the developers.

Software develops from baseline to baseline until it can be released. Changes to baselines must be controlled throughout the life cycle by documenting the:

- problems, so that they can be understood;
- changes that are proposed to solve the problem;
- modifications that result.

Problems that initiate the change process can result from bugs in code, faults in design, errors in the requirements or omissions in the requirements.

The change process must be monitored. Accurate records and reports of configuration item status are needed to provide visibility of controlled change.

Verification of the completeness and correctness of configuration items (by reviews and audits) is classified by ESA PSS-05-0 as a software verification and validation activity, and is discussed in ESA PSS-05-10, Guide to Software Verification and Validation'.

ESA PSS-05-0 identifies five primary configuration management activities:

- configuration identification;
- configuration item storage;
- configuration change control;
- configuration status accounting;
- release.

Figure 2.2.B shows the relationship between the software configuration management activities (shaded) and component development, system integration and build activities (unshaded). The diagram applies to all CIs, both documents and code, and shows the change control loop, a primary feature of software configuration management.

The following subsections discuss the principles of configuration management in terms of the five software configuration management activities. Software configuration management terms are collected in Appendix A for reference.

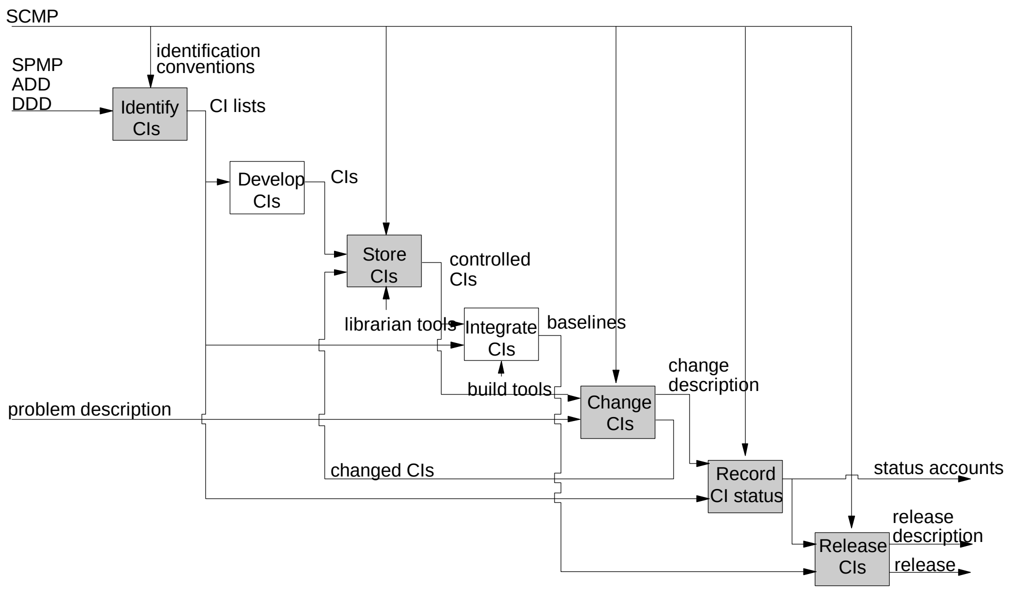

Figure 2.2B: Software Configuration Management Activities

2.2.1 Configuration identification

The first step in configuration management is to define the CIs and the conventions for their identification.

The inputs to the configuration identification activity are the SPMP (which lists the project documents), the ADD and the DDD (which list the software components). These inputs are used to define the hierarchy of CIs. The output is a configuration item list, defining what will be built using the naming conventions described in the SCMP. The configuration item list physically defines the components to be developed and the baselines to be integrated. For example an SPMP will say that an SRD will be written, but the configuration item list will define its file specification.

Configuration identification conventions should state how to:

- name a CI;
- decide who is the control authority of a CI;
- describe the history of a CI.

2.2.1.1 What to identify as configuration items

At the top level, the whole system is a CI. The system CI is composed of lower level CIs, which are derived from the design and planning documentation. Several factors may be relevant in deciding what to identify as the lowest level CI, such as the:

- software design defined in the ADD and DDD (the lowest level of the software design sets the lowest possible level of configuration management);
- capabilities of the software development tools (e.g. the units that the compiler or linkers input and output);
- bottom-level work packages defined in the SPMP.

Configurations must be practical from the physical point of view (i.e. each CI is easy to create and modify) and practical from the logical point of view (i.e. the purpose of each CI is easy to understand). A configuration should have sufficient 'granularity' (i.e. the lowest level CIs are small enough), to ensure that precise control is possible.

CIs should be easy to manage as discrete physical entities (e.g. files), and so the choice of what to make CIs can depend very much upon the tools available. A basic configuration management toolset (see Chapter 3) might rely on the file management facilities of the operating system. It is no accident therefore that CIs are often discrete files.

2.2.1.1.1 Baselines

A 'baseline' is a CI that has been formally reviewed and agreed upon (i.e. approved), and declared a basis for further development. Although any approved CI in a system can be referred to as a baseline, the term is normally used for the complete system.

Software development proceeds from baseline to baseline, accumulating new or revised CIs in the process. Early baselines contain only documents specifying the software to be built; later baselines contain the code as well.

The CIs in a baseline must be consistent with each other. Examples of consistency are:

- code CIs are compatible with the design document CIs;

- design document CIs are compatible with the requirements document CIs;
- user manual CIs describe the operation of the code CIs.

Baselines should be established when sharing software between developers becomes necessary. As soon as more than one person is using a piece of software, development of that software must be controlled.

Key baselines are tied to the major milestones in the life cycle (i.e. UR/R, SR/R, AD/R, DD/R, provisional acceptance and final acceptance). Baselines are also tied to milestones during integration. Unit-tested software components are assembled and integration-tested to form the initial baseline. New baselines are generated as other software components are integrated. The baselines provide a development and test environment for the new software components undergoing development and integration.

2.2.1.1.2 Variants

The term 'variant' is often used to identify CIs that, although offering almost identical functionality, differ in aspects such as:

- hardware environment;
- communications protocol;
- user language (e.g. English, French).

Variants may also be made to help diagnose software problems. The usefulness of such variants is often temporary, and they should be withdrawn when the problem has been solved.

The need to develop and maintain several variants can greatly complicate configuration management, so the number of variants should be minimised. One way to do this is to include conditional code in modules, so that one module can handle every variation. However modules can become very much more complicated, and developers need to trade-off code complexity with configuration complexity.

2.2.1.2 How to identify configuration items

Each configuration item must possess an identifier (SCM06). The identifier contains the name, type and version attributes of the configuration item (SCM07, SCM08, SCM09). The configuration management system should automatically ensure that all identifiers are unique.

Good names help developers quickly locate parts they need to work on, and ease traceability. Documents should have standard names (note that ESA PSS-05-0 predefines the names of documents) and code items should be named after the software components they contain.

The type field of a CI identifier should identify the processing the CI is intended for. There are three main types of CI:

- source CIs;
- derived CIs;
- tools to generate derived CIs from source CIs.

Source CIs (e.g. documents, source code) are created by software engineers. Corrections and modifications to CIs should always be done at source level.

Derived CIs (e.g. object files, executable files) are generated by processing source CIs with tools (e.g. compilers and linkers). The type field of a CI identifier should allow easy distinction between source CIs, derived CIs, and the tools used to make derived CIs.

Version numbers allow changes to CIs to be distinguished. Whenever a CI is changed, the version number changes. In most configuration management systems the version number is an integer.

Some configuration management systems distinguish between 'versions' and 'revisions'. Revision numbers track minor changes that do not change the functionality, such as bug fixes. When this approach is used, the version number actually consists of two numbers. The first marks major changes and the second, the revision number, marks minor changes. For example a software release might be referred to as 'Version 5.4'. For documents, the first number is called the 'issue' number. A version of a document might be referred to as 'Issue 2 Revision 1', for example.

The configuration identification method must accommodate new CIs, without requiring the modification of the identifiers of any existing CIs (SCM11). This is the 'extensibility' requirement. A method that sets a limit on the number of CIs does not meet the extensibility requirement, for example. When the limit has been reached, existing CIs have to be merged or deleted to make room for new CIs.

2.2.1.3 Configuration item control authority

Every CI should have a unique control authority that decides what changes will be made to a CI. The control authority may be an individual or a group of people. Three levels of control authority are normally distinguished in a software project:

- software author;
- software project manager;
- review board.

Large projects may have more levels. As a CI passes through the stages of verification (formal review meeting in the case of documents, unit, integration, system and acceptance tests in the case of code), so the control authority changes to match the higher verification status of the CI.

2.2.1.3.1 Software author

Software authors create low-level CIs, such as documents and code. Document writers are usually the control authority for draft documents. Programmers are normally the control authority for code until unit testing is complete.

2.2.1.3.2 Software project manager

Software project managers are responsible for the assembly of high-level CIs (i.e. baselines and releases) from the low-level CIs provided by software authors. Software project managers are the control authorities for all these CIs. Low-level CIs are maintained in software libraries (see Section 2.2.2.1). On most projects the software project manager is supported by a software librarian.

2.2.1.3.3 Review board

The review board approves baselines and changes to them. During development phases, formal review of baselines is done by the UR/R, SR/R, AD/R and DD/R boards. During the TR and OM phases, baselines are controlled by the Software Review Board (SRB).

The review board is the control authority for all baselines that have been approved or are under review. The review board may decide that a baseline should be changed, and authorises the development or maintenance team to implement the change. Control authority rights are not returned. Those responsible for implementing changes must resist the temptation to make any changes that have not been authorised. New problems found should be reported in the normal way (see Sections 2.3.3 and 2.4.3).

Review boards should be composed of members with sufficient authority and expertise to resolve any problems with the software. The nature of the review board depends upon the project. On a small project it might just consist of two people, such as the project manager and a user representative. On a large project the membership might include the project manager, the technical manager, the software librarian, the team leaders, user representatives and quality assurance personnel. Review board procedures are discussed in ESA PSS-05-10, 'Guide to Software Verification and Validation'.

2.2.1.4 Configuration item history

The development history of each CI must be recorded from the moment it is first submitted for review or integration. For documents, this history is stored in Document Change Records (DCRs) and Document Status Sheets (DSS). For source code, the development history is stored in the module header (SCM18). Change records in module headers should reference the Software Change Request (SCR) that actioned them. For derived code, the development history is stored in the librarian's database. The development history should include records of the tests that have been performed. The Software Modification Report (SMR) should summarise the changes to CIs that have been revised in response to an SCR.

Development histories may be stored as tables of 'derivation records'. Each derivation record records one event in the life of the CI (e.g. source code change, compilation, testing etc.).

2.2.2 Configuration item storage

All CIs must be stored securely so that they never get lost. Software projects can accumulate thousands of low-level CIs, and just as books are stored in libraries, it is necessary to store low-level CIs in software libraries. The software libraries are themselves CIs.

Software CIs reside on hardware media (e.g. paper, magnetic disk). Storage media must be maintained securely and safely so that software is never lost or its integrity compromised.

2.2.2.1 Software libraries

A software library is a controlled collection of configuration items (e.g. source module, object module and document). Low-level CIs are developed and stored in libraries (e.g. source and object libraries), and then extracted to build high-level CIs (e.g. executable images).

As a minimum, every project must use the following software libraries to store CIs:

- development (or dynamic) libraries (SCM23);
- master (or controlled) libraries (SCM24);
- archive (or static) libraries (SCM25).

Development libraries store CIs that are being written or unit tested. Master libraries store CIs in the current baselines. Archive libraries store CIs in releases or retired baselines. CIs in archive libraries must not be modified (SCM26).

Development libraries are created and maintained by the software authors in their own work space. Master and archive libraries are created and maintained by the software librarian, who is responsible for:

- establishing new master libraries;
- updating master libraries;
- backup of master libraries to archive libraries;
- control of access to master libraries and archive libraries.

To make Figure 2.2B simple, the only storage activity shown is that of the software librarian storing new and changed CIs in master libraries. CIs are also stored in development libraries during development and change activities. CIs are stored in archive libraries when the software is released or routinely saved. In Figure 2.2B the outputs of the storage activity are the controlled CIs which are input to the integration process, or to the software change process.

Software libraries are fundamental to a software configuration management system. Whatever the library, the software configuration management system should permit CIs in libraries to be read, inserted, replaced and deleted.

Access to software libraries should be controlled so that it is impossible:

- to access CIs without the correct authorisation;
- for two people to simultaneously update the same CI.

Tables 2.2.2.1A and B show the software library access rights that should be granted to development staff and software librarians. Read access allows a CI to be examined or copied. The insert access right allows a new CI to be added to a software library. The delete access right allows a CI to be removed from a library. The replace right allows a CI to be removed from a library and another version inserted.

library \ access right	**read**	**insert**	**replace**	**delete**
development libraries	y	y	y	y
master libraries	y	n	n	n
archive libraries	y	n	n	n

Table 2.2.2.1A: Development staff access rights

library \ access right	**read**	**insert**	**replace**	**delete**
development libraries	y	y	y	y
master libraries	y	n	n	n
archive libraries	y	n	n	n

Table 2.2.2.1B: Software librarian access rights

Table 2.2.2.1A does not mean that a development library should be accessible to all developers. On the contrary, apart from the software librarian, only the person directly responsible for developing or maintaining the CIs in a development library should have access to them. For sharing software, master libraries should be used, not development libraries.

Simultaneous update occurs when two or more developers take a copy of a CI and make changes to it. When a developer returns a modified CI to the master library, modifications made by developers who have returned their CI earlier are lost. A charge-in/charge-out or 'locking' mechanism is required to prevent simultaneous update.

2.2.2.2 Media control

All CIs should be stored on controlled media (e.g. tapes, disks). Media should be labelled, both visibly (e.g. sticky label) and electronically (e.g. tape header). The label should have a standard format, and must contain the:

- project name (SCM19);
- configuration item identifier (name, type, version) (SCM20);
- date of storage (SCM21);
- content description (SCM22).

Procedures for the regular backup of development libraries must be established (SCM28). Up-to-date security copies of master and archive libraries must always be available (SCM27). An aim of good media control is to be able to recover from media loss quickly.

2.2.3 Configuration change control

Changes are normal in the evolution of a system. Proper change control is the essence of good configuration management.

Software configuration control evaluates proposed changes to configuration items and coordinates the implementation of approved changes. The change control process (see Figure 2.2B) analyses problem descriptions (such as RIDs and SPRs), decides which (if any) controlled CIs are to be changed, retrieves them from the master library for change, and returns changed CIs to the master library for storage. The process also outputs descriptions of the changes.

The change process is a miniature software development itself. Problems must be identified, analysed and a solution designed and implemented. Each stage in the process must be documented. ESA PSS-05-0 describes procedures for changing documents and code that are part of a baseline. These procedures are discussed in detail in Section 2.3.3 and 2.4.3.

Changes must be verified before the next changes are implemented. If changes are done in a group, each change must be separately verifiable. For documents, this just means that the Document

Change Record (DCR) at the beginning of a document should describe each change precisely, or that a revised document is issued with change bars. For code, groups of changes must be carefully analysed to eliminate any confusion about the source of a fault during regression testing. If time permits, changes are best done one at a time, testing each change before implementing the next. This 'one-at-a-time' principle applies at all levels: code modification and unit testing; integration and integration testing, and system testing.

If an evolutionary or incremental delivery life cycle approach is used, changes may be propagated backwards to releases in use, as well as forwards to releases under development. Consider two releases of software, 1 and 2. Release 1 is in operational use and release 2 is undergoing development. As release 2 is developed, release 1 may evolve through several baselines (e.g. release 1.0, 1.1, 1.2 etc.) as problems discovered in operation are fixed.

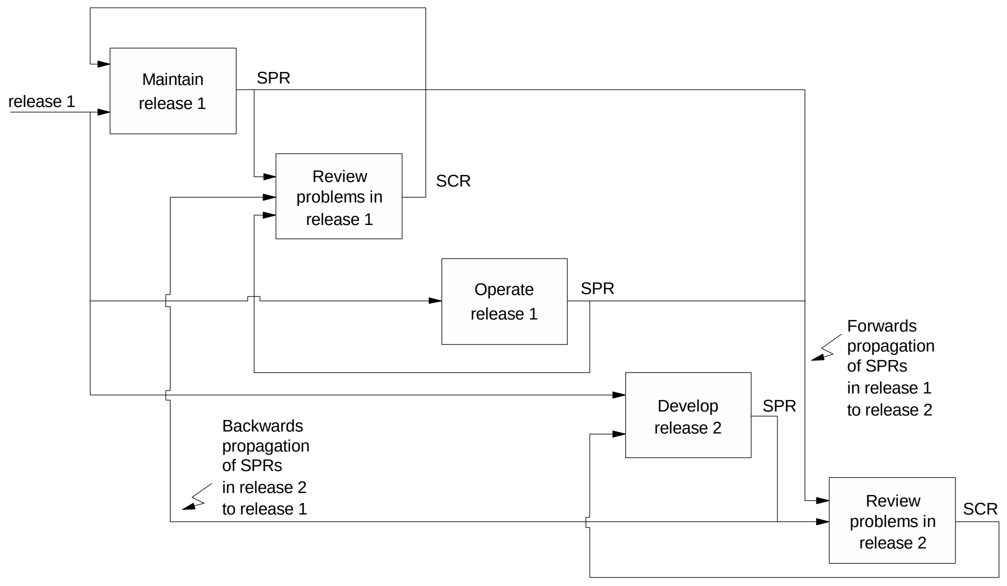
Maintain release 1
release 1
SPR
Review problems in release 1
SCR
Operate release 1
SPR
Forwards propagation of SPRs in release 1 to release 2
Develop release 2
SPR
Backwards propagation of SPRs in release 2 to release 1
Review problems in release 2
SCR

Figure 2.2.3: Software problem report propagation

Reports of problems found in release 1 must be fed forwards to the developers of release 2. Reports of problems in release 2 must also be fed backward to the maintainers of release 1, so that they can decide if the problems need to be fixed in release 1. Figure 2.2.3 shows how this can be done. The maintainers of release 1 should consider whether the problems pose a hazard to the users of release 1, and update it if they do.

2.2.4 Configuration status accounting

Software configuration status accounting is 'the recording and reporting of information needed to manage a configuration effectively, including a listing of the approved configuration identification, the status of proposed changes to the configuration, and the implementation status of the proposed changes' [Ref. 2]. The status of all configuration items must be recorded (SCM31). Configuration status accounting continues throughout the life cycle.

In Figure 2.2.B, the Record CI Status process examines the configuration item lists output by the Identify CIs process, and the change descriptions output by the Change CIs process, to produce the status accounts.

The software librarian is normally responsible for maintaining the configuration status accounts, and for checking that the 'as-built' configuration status complies with the 'as-designed' status defined in the SCMP.

Configuration status accounts are composed of:

- records that describe the contents, dates and version/issue of each baseline (SCM32 and SCM35);
- records that describe the status of problem reports and their solutions (SCM33 and SCM34).

Configuration status accounts can be presented as tables of these two types of records.

Table 2.2.4 is an example of a table of baseline records. Each column of the table lists CIs in a baseline. The table shows the issue and revision number of each document, and the version number of the code in seven baselines. Six of the baselines are associated with the six major milestones of the software project. An extra interim baseline is created during integration. Table 2.2.4 assumes that revisions to each document are required between each baseline to show how the version numbers change. This is an extreme case.

Each management document (i.e. SPMP, SCMP, SVVP, SQAP) is reissued at each major milestone. This corresponds to the inclusion of new sections describing the plans for the next phase.

Baseline	**1**	**2**	**3**	**4**	**5**	**6**	**7**
CI\Milestone	**URD approval**	**SRD approval**	**ADD approval**	**Interim Baseline**	**DDD approval**	**Prov' Accept '**	**Final Accept '**
SPMP	1.0	2.0	3.0	4.0	4.0	4.1	4.2
SCMP	1.0	2.0	3.0	4.0	4.0	4.0	4.0
SVVP	1.0	2.0	3.0	4.0	4.1	4.2	4.3
SQAP	1.0	2.0	3.0	4.0	4.0	4.0	4.0
URD issue	1.0	1.1	1.2	1.3	1.4	1.5	1.6
SRD issue		1.0	1.1	1.2	1.3	1.4	1.5
ADD issue			1.0	1.1	1.2	1.3	1.4
DDD issue				1.0	1.1	1.2	1.3
SUM issue				1.0	1.1	1.2	1.3
Program A				1.0	1.1	1.2	1.3
Program B				1.0	1.1	1.2	1.3
Compiler				5.2	5.2	5.2	5.2
Linker				3.1	3.1	3.1	3.1
Oper. Sys				6.1	6.1	6.1	6.1
Program C					1.0	1.1	1.2
STD issue						1.0	1.0
SRD issue							1.0

Table 2.2.4: Example of baseline records

Records should be kept of the progress of Software Problem Reports (SPR) and Review Item Discrepancies (RID). A record should be created for each RID and SPR when it is received.

Each record should contain fields for noting:

- whether the SPR or RID has been approved;
- the corresponding SCR, SMR or RID.

The records do not actually describe the changes - this is what the CI history does (see Section 2.2.1.4).

Configuration status accounts are important instruments of project control, and should be made available to all project members for up-to-date information on baseline status. When a program works one day and not the next, the first question is 'what has changed?'. The configuration status accounts should answer such questions quickly and accurately. It should be possible to reproduce the original baseline, verify correct operation, and individually reintroduce the changes until the problem recurs [Ref. 6].

Values of quality metrics (e.g. reliability) might be derived by means of data in the configuration status accounts, and these metrics might be used to decide the acceptability of the software. Sometimes quite simple pieces of information can be highly significant. A module that requires frequent 'bug fixes' may have more serious structural problems, and redesign may be necessary.

Sections 2.3.4 and 2.4.4 discuss the details of the configuration status accounting of documents and code.

2.2.5 Release

When a CI is distributed outside the project it is said to be 'released'. In Figure 2.2B, inputs to the release process are the baselines that have been successfully system tested, and the status accounts (for deciding the readiness for release). Outputs of the release process are the release itself and its 'release description'.

The Software Transfer Document (STD) describes the first release. The configuration item list in the first release must be included in the STD (SCM12). A Software Release Note (SRN) describes subsequent releases of the software (SCM14). A list of changed configuration items must be included in each SRN (SCM13).

2.3 DOCUMENTATION CONFIGURATION MANAGEMENT

ESA PSS-05-0 requires several documents to be produced in the software life cycle. The documents may specify the product (e.g. Software Requirements Document, Architectural Design Document) or some part of the process (e.g. Software Verification and Validation Plan, Software Configuration Management Plan). Whatever the document, it must be subject to configuration management procedures (SCM01).

The following subsections discuss the configuration management functions from the point of view of documentation.

2.3.1 Document configuration identification

A document CI may be:

- an entire document;
- a part of a document (e.g. a diagram).

The general principle established in ESA PSS-05-0 is that any configuration identification method must accommodate new CIs without modifying any existing CIs (SCM11). This principle applies to documents and parts of documents just as much as to code.

For documentation, the CI identifier must include:

- a number or name related to the purpose of the CI (SCM07);
- an indication of the type of processing for which the CI is intended (SCM08);
- an issue number and a revision number (SCM10).

As an example, the identifier of Issue 1, Revision 3 of the Software Requirements Document for a project called XXXX, produced with a word processor that attaches the filetype DOC, might be:

XXXX/SRD/DOC/Issue 1/Revision 3

The DOC field relates to SCM08, the Issue and Revision fields contain the version information required by SCM10, and the remaining fields name the item, as required by SCM07.

The title page(s) of a document should include:

- project name (SCM19);
- configuration item identifier (name, type, version) (SCM20);
- date (SCM21);
- content description (SCM22) (i.e. abstract and table of contents).

2.3.2 Document configuration item storage

ESA PSS-05-0 requires that all CIs be stored in software libraries. Paper versions of documents should be systematically filed in the project library. Electronic documents should be managed with the aid of software configuration management tools and should be stored in electronic libraries.

2.3.3 Document configuration change control

The procedure for controlling changes to a document is shown in Figure 2.3.3. This diagram shows the decomposition of the Change CIs' process in Figure 2.3.3.B. The process is used when a document that is part of a major baseline is modified, and is a mandatory requirement of ESA PSS-05-0 (SCM29).

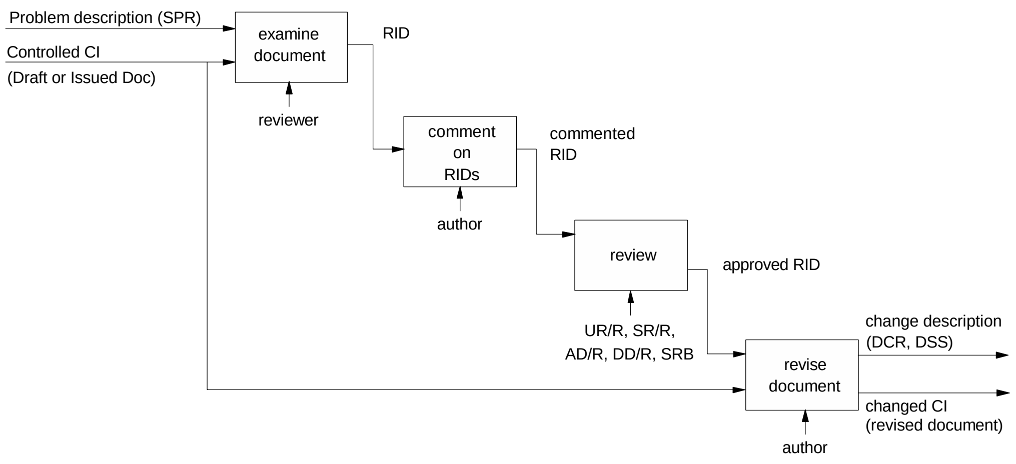

Figure 2.3.3: Document configuration change control activities

Examination is the first step in the document release process. Copies of the documents are circulated to everyone concerned (who should be identified in the Software Verification and Validation Plan). In the DD, TR and OM phases, SPRs may be input to the examination stage, to guide reviewers looking for errors in the URD, SRD, ADD, DDD and SUM.

Reviewers describe the defects that they find in the document on Review Item Discrepancy (RID) forms. RIDs permit reviewers to describe the:

- CI identifier of the document;
- problem location (in terms of a lower level CI identifier, if appropriate);
- problem;
- possible solution recommended by the reviewer.

RIDs are returned to the document authors for comment. Authors need to study problems identified in RIDs prior to the review meeting and mark their response on the RIDs, recording their views on the problems and the recommended solutions. Their responses may include preliminary assessments of the resources required to make the recommended changes, and the risks involved.

The commented RIDs and draft document are then input to the formal review meeting. Procedures for formal review meetings are discussed in ESA PSS-05-10, Guide to Software Verification and Validation. The review meeting may be a UR/R, SR/R, AD/R, DD/R or SRB meeting, depending upon the phase of the project.

The review meeting should discuss the RIDs and decide whether to:

- close the review item because the update has been completed;
- update (i.e. change) the document on the basis of the review item comments;
- reject the review item;
- define actions needed before a decision to close, update or reject can be made.

Each meeting should consider the level of authority required to approve each update. Agreements with other organisations may be necessary before decisions on some RIDs can be made.

The review meeting ends with a decision on whether to approve the document. A document may be approved although some changes have been agreed. Until the document has been updated, the baseline consists of the examined document and the RIDs describing the updates. Such RIDs are normally put in an appendix to the document.

A new version of the document incorporating the updates is then prepared. All updates should be recorded in Document Change Records (DCRs) and on the Document Status Sheet (DSS). Because the updates have the authority of the review board, authors must implement update instructions in the RIDs exactly.

Examples of RIDs, DCRs and DSSs are contained in Appendix D.

Sometimes a project finds that so many changes to a document are required that the RID method of recording problems is too clumsy. In this situation reviewers may prefer to:

- mark up a copy of the document;
- describe the problems in a note.

Review by these methods is acceptable only when the first issue of a document is being prepared, because there is no baseline for comparison until the first issue appears. After the first issue, the RID/DCR/DSS method must be used, because it allows the accurate tracking of changes to baselines, which the other methods do not.

2.3.4 Document configuration status accounting

Configuration status accounting keeps track of the contents of baselines. The configuration status accounts should describe the issue and revision number of each document in every baseline (see Section 2.2.4).

The configuration status accounts must record the status of every RID outstanding against every document in the baseline (SCM33). Table 2.3.4 describes the configuration status accounts of the RIDs in baseline 3 in Table 2.2.4.

RID Number	CI	Date submitted	Date reviewed	Status	DCR Number	CI with DCR
1	URD 1.1	1/6/92	1/7/92	closed	1	URD 1.2
2	URD 1.1	5/6/92	1/7/92	rejected	-	-
3	URD 1.1	6/6/92	1/7/92	action		
4	SRD 1.0	7/6/92	2/7/92	update		

Table 2.3.4: Records of the status RIDs in baseline 3

It is also important to track the RIDs outstanding against documents. Configuration status accounts need to be organised in a variety of ways (e.g. CI identifier order) and a DBMS is useful for managing them.

2.3.5 Document release

A document is released when it is formally 'issued'. The Document Status Sheet (DSS) is updated to mark this event, and any DCRs (remaining from previous revisions) are omitted. There should be no outstanding RIDs when a document is issued.

A formal issue must carry the approval of the responsible review body. Only formally issued documents can be made applicable. Approvals should be marked on the title page.

2.4 CODE CONFIGURATION MANAGEMENT

Source code, object code, executable code, data files, tools, test software and data must be subjected to configuration management procedures (SCM01). Procedures for the configuration management of code need to be defined before any code is written. This may be as early as the

UR phase if prototypes are needed. DD phase code configuration management procedures need to be defined at the end of the AD phase in the SCMP/AD. These procedures may differ very much from those used in previous phases.

2.4.1 Code configuration identification

A code CI may be:

- a data file;
- an executable program;
- an object module;
- a source module.

The CI name must represent its function or specific purpose (SCM07). Since the name used for a software component in the ADD or DDD must also summarise the function of the component, the software component name should be used in the configuration identifier. The name of a source module that calculates an average might be called CALC_MEAN', for example. This configuration identification method allows easy traceability between the design and the CIs that implement it.

The CI type must summarise the processing for which the item is intended. If CALC_MEAN is a FORTRAN source file, its type would be written as FOR'. The configuration identifier of the first version of the file is then CALC_MEAN.FOR.1'.

The identifier of a CI must distinguish it from other items with different:

- requirements, especially functionality and interfaces (e.g. CALC_ARITHMETIC_MEAN and CALC_GEOMETRIC_MEAN) (SCM04);
- implementation (e.g. CALC_MEAN coded in FORTRAN and CALC_MEAN coded in C) (SCM05).

Source modules must have a header that includes:

- configuration item identifier (name, type, version) (SCM15);
- author (SCM16);
- creation date (SCM17);
- change history (version/date/author/description) (SCM18).

Projects should define a standard module header in the DDD. The format of the header must follow the rules of the selected programming language. Configuration management tools and documentation tools may be built to handle header information.

Code configuration identification conventions need to be established before any design work is done. They should be strictly observed throughout the design, coding, integration and testing.

2.4.2 Code configuration item storage

Librarians are perhaps the most widely used software development tools after editors, compilers and linkers. Librarian tools are used to:

- store source and object modules;
- keep records of library transactions;
- cross-reference source and object modules.

Once a source code module has been created and successfully compiled, source and object code are inserted into the development libraries. These libraries are owned by the programmer.

Programmers must observe configuration management procedures during coding and unit testing. The CI identifiers for the modules stored in their development libraries should be the same as the identifiers for the corresponding modules in master libraries. This makes CI status easy to track. Programmers should not, for example, write a module called CM.FOR and then rename it CALC_MEAN.FOR when it is submitted for promotion to the master library.

Code should be portable. Absolute file references should not be hardwired in source modules. Information relating names in software to names in the environment should be stored in tables that can be easily accessed.

2.4.3 Code configuration change control

The procedure for controlling changes to released code is shown in Figure 2.4.3. This diagram is a decomposition of the Change CIs' process in Figure 2.2B. The process is used when code that is part of a baseline is modified, and is a mandatory requirement of ESA PSS-05-0 (SCM30).

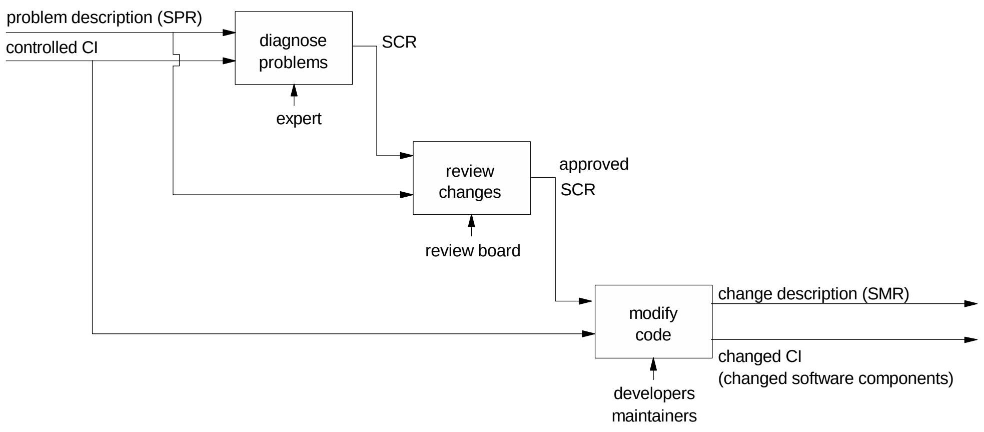

Figure 2.4.3: Code configuration change control activities

Problems may be identified during integration, system testing, acceptance testing or operations by users, development staff and quality assurance personnel. Software Problem Reports (SPRs) contain:

- CI title or name;
- CI version or release number;
- priority of the problem with respect to other problems;
- a description of the problem;
- operating environment;
- recommended solution (if possible).

The SPR is then assigned to an expert for diagnosis. Some debugging of code is often required, and the expert will need to have access to all the software in the baseline. The expert must prepare a Software Change Request (SCR) if modifications to the software are required. The SCR describes the:

- CI title or name;
- CI version or release number;
- priority of the problem with respect to other problems;
- changes required;
- responsible staff;
- estimated start date, end date and manpower effort.

The review board (i.e. the DD/R, SRB or a delegated individual) then reviews each SPR and associated SCRs and decides whether to:

- close the SPR because the update has been completed;
- update (i.e. change) the software on the basis of the
- recommendations in the associated SCR;
- reject the SPR;
- define an action that needs to be taken before a decision to close, update or reject can be made.

Approved SCRs are passed to the developers or maintainers for implementation, who produce a Software Modification Report (SMR) for each SCR. An SMR must describe the:

- CI title or name;
- CI version or release number;
- changes implemented;
- actual start date, end date and manpower effort.

Attachments may accompany the SPR, SCR and SMR to detail the problem, the change and the modification.

The code to be changed is taken from the master library used to build the baseline that exhibited the problem. Regression testing is necessary when changes are made (SCM39).

2.4.4 Code configuration status accounting

Configuration status accounting keeps track of the contents of baselines, allowing software evolution to be followed, and old baselines to be reconstructed.

Records of the status of each CI should be kept. These should describe whether the CI has passed project milestones, such as unit testing and integration testing. The table of these records provides a snapshot of baseline status, and provides management with visibility of product development.

The configuration status accounts must record the status of every SPR, SCR and SMR related to every CI in the baseline (SCM34). Table 2.4.4 shows how such records might be kept.

CI Name Version	SPR	Date submitted	Review Date	SPR Decision	Related SCR	Related SMR	Completion Date
LOADER v1.1	2	5/8/92	1/9/92	update	2	2	1/10/92
LOADER v1.1	3	7/8/92	1/9/92	update	3	2	1/10/92
EDITOR v1.2	4	9/8/92	1/9/92	rejected	-	-	-
MERGER v1.0	1	1/8/92	1/9/92	update	1	1	3/10/92

Table 2.4.4: Configuration status accounts example

2.4.5 Code release

The first release of the code must be documented in the STD. Subsequent releases must be accompanied by a Software Release Note (SRN). STDs and SRNs provide an overview of a release. They include a configuration item list that catalogues the CIs in the release. They must summarise any faults that have been repaired and the new requirements that have been incorporated (SCM36). One way to do this is to list the SPRs that have been dealt with.

For each release, documentation and code must be consistent (SCM37). Furthermore, old releases must be retained, for reference (SCM38). Where possible, the previous release should be kept online during a change-over period, to allow comparisons, and as a fallback. The number of releases in operational use should be minimised.

Releases should be self-identifying (e.g. label, display or printed output) so that users know what the software is. Some form of software protection may be desirable to avoid unauthorised use of a release.

Modified software must be retested before release (SCM29). Tests should be selected from the SVVP to demonstrate its operational capability.

CHAPTER 3
SOFTWARE CONFIGURATION MANAGEMENT TOOLS

3.1 INTRODUCTION

Software configuration management is often described as simple in concept but complex in detail. Tools that support software configuration management are widely available and their use is strongly recommended.

This chapter does not describe particular tools. Instead, this chapter discusses software configuration management tools in terms of their capabilities. No single tool can be expected to provide all of them, and developers need to define their own software configuration management tool requirements and assemble a toolset' that meets them.

ANSI/IEEE Std 1042-1987, IEEE Guide to Software Configuration Management, recognises four types of toolset:

- basic;
- advanced;
- online;
- integrated.

This classification is used to organise software configuration management tools. Each toolset includes the capabilities of preceding toolsets.

3.2 BASIC TOOLSET REQUIREMENTS

Basic toolsets use standard operating system utilities such as the editor and librarian system, perhaps supplemented by a database management system. They rely heavily on personal discipline. The capability requirements for a basic toolset are listed below.

1. A basic toolset should permit the structured definition, identification and storage of configuration items.

   The file systems of operating systems allows storage of CIs in files. A file system should object if the user tries to create a file with the same name as an existing file, i.e. it should ensure that names are unique.

2. A basic toolset should permit the structured storage of configuration items.

   The file systems of operating systems should allow the creation of directory trees for storing files.

3. A basic toolset should provide a librarian system that supports:
   - insertion of modules;
   - extraction of modules;
   - replacement of modules by updated modules;
   - deletion of modules;
   - production of cross-reference listings to show which modules refer to which;
   - storage of the transaction history;
   - all the above features for both source (i.e. text) and object (i.e. binary) files.

4. A basic toolset should provide facilities for building executable software from designated sources.

   This implies the capability to create a command file of build instructions with a text editor.

5. A basic toolset should provide security features so that access to CIs can be restricted to authorised personnel.

   This means that the operating system should allow the owner of a file to control access to it.

6. A basic toolset should provide facilities for comparing source modules so that changes can be identified.

   Most operating systems provide a tool for differencing ASCII files.

7. A basic toolset should include tools for the systematic backup of CIs. Specifically:
   - automatic electronic labelling of media;
   - complete system backup;
   - incremental backup;
   - restore;
   - production of backup logs.

   Most operating systems have commands for backing-up and restoring files.

3.3 ADVANCED TOOLSET REQUIREMENTS

Advanced toolsets supplement the basic toolset with standalone tools such as source code control systems and module management systems. The extra capability requirements for an advanced toolset are listed below.

1. An advanced toolset should provide a locking system with the library to ensure that only one person can work on a module at a time.

   Most operating system librarians do not provide this feature; it is a characteristic of dedicated software configuration management librarian tools.

2. An advanced toolset should minimise the storage space needed.

   One way to do this is store the latest version and the changes, usually called 'deltas', required to generate earlier versions.

3. An advanced librarian tool should allow long names to be used in identifiers.

   Some operating systems force the use of shorter names than the compiler or programming language standard allows, and this can make it impossible to make the configuration identifiers contain the names used in design.

4. An advanced toolset should permit rollback, so that software configuration management operations can be undone.

   A simple but highly desirable type of backtracking operation is the 'undelete'. Some tools can reverse deletion operations.

5. An advanced toolset should provide facilities for rebuilding executable software from up-to-date versions of designated sources.

   This capability requires the build tool to examine the dependencies between the modules in the build and recompile those modules that have been changed since the last build.

6. An advanced toolset should record the version of each module that was used to build a product baseline, so that product baselines can always be reproduced.

7. An advanced toolset should provide facilities for the handling of configuration status accounts, specifically:
   - insertion of new RID, SPR, SCR and SMR status records;
   - modification of RID, SPR, SCR and SMR status records;
   - deletion of RID, SPR, SCR and SMR status records;

- search of RID, SPR, SCR and SMR records on any field;
- differencing of configuration status accounts, so that changes can be identified;
- report generation.

3.4 ONLINE TOOLSET REQUIREMENTS

Online toolsets supplement the capabilities of advanced toolsets by providing facilities for interactive entry of change control information. The extra capability requirements for an online toolset are listed below.

1. An online toolset should provide facilities for the direct entry of RID, SPR, SCR and SMR information into the database;
2. An online toolset should provide an authentication system that permits online approval of changes.

3.5 INTEGRATED TOOLSET REQUIREMENTS

Integrated toolsets supplement the capabilities of online toolsets by extending the change control system to documentation. Integrated Toolsets form part of so-called 'Integrated Project Support Environments' (IPSEs), 'Project Support Environments' (PSEs) and 'Software Development Environments' (SDEs). The extra capabilities of an integrated toolset are listed below.

1. An integrated toolset should recognise all the dependencies between CIs, so that consistency can be controlled.

   This capability requires a super-library called a 'repository'. The Online Toolset has to be integrated with the CASE tools used for design. This facility permits the system to be automatically rebuilt from a design change, not just a code change.

2. An integrated toolset should have standard interfaces to other tools (e.g. project management tools).

   Standards for interfacing software tools have been proposed, such as the Portable Common Tools Environment (PCTE) [Ref. 7] and the Common APSE Interface Set (CAIS) [Ref. 8].

CHAPTER 4
THE SOFTWARE CONFIGURATION MANAGEMENT PLAN

4.1 INTRODUCTION

All software configuration management activities shall be documented in the Software Configuration Management Plan (SCM40). A new section of the SCMP must be produced for each development phase (SCM42, SCM44, SCM46, SCM48). Each SCMP section must document all software configuration management activities (SCM40), specifically the:

- organisation of configuration management;
- procedures for configuration identification;
- procedures for change control;
- procedures for configuration status accounting;
- tools, techniques and methods for software configuration management;
- procedures for supplier control;
- procedures for the collection and retention of records.

The size and content of the SCMP should reflect the complexity of the project. ANSI/IEEE Std 1042-1987, IEEE Guide to Software Configuration Management contains examples of plans for:

- a complex, critical computer system;
- a small software development project;
- maintaining programs developed by other activities or organisations;
- developing and maintaining embedded software.

Configuration management procedures must be in place before software production (code and documentation) starts (SCM41). Software configuration management procedures should be easy to follow and efficient. Wherever possible, procedures should be reusable in later phases. Instability in software configuration management procedures can impede progress in a software project.

4.2 STYLE

The SCMP should be plain and concise. The document should be clear, consistent and modifiable.

The author of the SCMP should assume familiarity with the purpose of the software, and not repeat information that is explained in other documents.

4.3 RESPONSIBILITY

The developer is responsible for the production of the SCMP.

4.4 MEDIUM

It is usually assumed that the SCMP is a paper document. There is no reason why the SCMP should not be distributed electronically to participants with the necessary equipment.

4.5 CONTENT

The SCMP is divided into four sections, one for each development phase. These sections are called:

- Software Configuration Management Plan for the SR phase (SCMP/SR);
- Software Configuration Management Plan for the AD phase (SCMP/AD);
- Software Configuration Management Plan for the DD phase (SCMP/DD);
- Software Configuration Management Plan for the TR phase (SCMP/TR).

ESA PSS-05-0 recommends the following table of contents for each section of the SCMP, which as been derived from the IEEE Standard for Software Configuration Management Plans (ANSI/IEEE Std 828-1990).

Service Information:

a - Abstract
b - Table of Contents
c - Document Status Sheet
d - Document Change records made since last issue

1 INTRODUCTION
   1.1 Purpose
   1.2 Scope
   1.3 Glossary
   1.4 References

2 MANAGEMENT

3 CONFIGURATION IDENTIFICATION

4 CONFIGURATION CONTROL
   4.1 Code (and document) control
   4.2 Media control
   4.3 Change control

5 CONFIGURATION STATUS ACCOUNTING

6 TOOLS TECHNIQUES AND METHODS FOR SCM

7 SUPPLIER CONTROL

8 RECORDS COLLECTION AND RETENTION

Material unsuitable for the above contents list should be inserted in additional appendices. If there is no material for a section then the phrase Not Applicable' should be inserted and the section numbering preserved.

4.5.1 SCMP/1 INTRODUCTION

The following subsections should provide an introduction to the plan.

4.5.1.1 SCMP/1.1 Purpose

This section should:

(1) briefly define the purpose of the particular SCMP;

(2) specify the intended readership of the SCMP.

4.5.1.2 SCMP/1.2 Scope

This section should identify the:

(1) configuration items to be managed;

(2) configuration management activities in this plan;

(3) organisations the plan applies to;

(4) phase of the life cycle the plan applies to.

4.5.1.3 SCMP/1.3 Glossary

This section should define all terms, acronyms, and abbreviations used in the plan, or refer to other documents where the definitions can be found.

4.5.1.4 SCMP/1.4 References

This section should provide a complete list of all the applicable and reference documents, identified by title, author and date. Each document should be marked as applicable or reference. If appropriate, report number, journal name and publishing organisation should be included.

4.5.2 SCMP/2 MANAGEMENT

This section should describe the organisation of configuration management, and the associated responsibilities. It should define the roles to be carried out. ANSI/IEEE Std 828-1990, Standard for Software Configuration Management Plans' [Ref. 3] recommends that the following structure be used for this section.

4.5.2.1 SCMP/2.1 Organisation

This section should:

- identify the organisational roles that influence the software configuration management function (e.g. project managers, programmers, quality assurance personnel and review boards);
- describe the relationships between the organisational roles;
- describe the interface with the user organisation.

Relationships between the organisational roles may be shown by means of an organigram. This section may reference the SPMP.

4.5.2.2 SCMP/2.2 SCM responsibilities

This section should identify the:

- software configuration management functions each organisational role is responsible for (e.g. identification, storage, change control, status accounting);

- responsibilities of each organisational role in the review, audit and approval process;
- responsibilities of the users in the review, audit and approval process.

4.5.2.3 SCMP/2.3 Interface management

This section should define the procedures for the management of external hardware and software interfaces. In particular it should identify the:

- external organisations responsible for the systems or subsystems with which the software interfaces;
- points of contact in the external organisations for jointly managing the interface;
- groups responsible for the management of each interface.

4.5.2.4 SCMP/2.4 SCMP implementation

This section should establish the key events in the implementation of the SCMP, for example the:

- readiness of the configuration management system for use;
- establishment of the Software Review Board;
- establishment of baselines;
- release of products.

The scheduling of the software configuration management resources should be shown (e.g. availability of software librarian, software configuration management tools and SRB). This section may cross-reference the SPMP.

4.5.2.5 SCMP/2.5 Applicable policies, directives and procedures

This section should:

- identify all applicable software configuration management policies, directives or procedures to be implemented as part of this plan (corporate software configuration management documents may be referenced here, with notes describing the parts of the documents that apply);

- describe any software configuration management polices, directives or procedures specific to this project, for example:
  - project-specific interpretations of corporate software configuration management documents;
  - level of authority required for each level of control;
  - level of review, testing or assurance required for promotion.

4.5.3 SCMP/3 CONFIGURATION IDENTIFICATION

This section should describe the conventions for identifying the software items and then define the baselines used to control their evolution.

4.5.3.1 SCMP/3.1 Conventions

This section should:

- define project CI naming conventions;
- define or reference CI labelling conventions.

4.5.3.2 SCMP/3.2 Baselines

For each baseline, this section should give the:

- identifier of the baseline;
- contents, i.e. the:
  - software itself (e.g. URD, SRD, ADD, DDD, modules, executables, SUM, SVVP);
  - tools for making derived items in the baseline (e.g. compiler, linker and build procedures);
  - test software (e.g. data, harnesses and stubs);
  - RIDs, SPRs etc that relate to the baseline;
- ICDs, if any, which define the interfaces of the software;
- review and approval events, and the acceptance criteria, associated with establishing each baseline;
- participation required of developers and users in establishing baselines.

Because the SCMP is a plan, the precise contents of each baseline may not be known when it is written (e.g. names of modules before the detailed design is started). When this occurs, procedures for getting a report of the contents of the baseline should be defined (e.g. a directory list). An example of a report may be supplied in this section.

If appropriate, the description of each baseline should:

- distinguish software being developed from software being reused or purchased;
- define the hardware environment needed for each configuration;
- trace CIs to deliverable items listed in the SPMP, and, for code, to the software components described in the ADD and DDD.

4.5.4 SCMP/4 CONFIGURATION CONTROL

In ANSI/IEEE Std 610.12-1990 [Ref. 2], configuration control covers only configuration item change control. In ESA PSS-05-0, the definition of configuration control is expanded to include configuration item storage. Sections 4.1, Code Control' and 4.2, Media Control' therefore describe the procedures for configuration item storage. Section 4.3 of the plan describes the procedures for configuration item change control.

4.5.4.1 SCMP/4.1 Code (and document) control

This section should describe the software library handling procedures. ESA PSS-05-0 calls for three types of library:

- development (or dynamic);
- master (or controlled);
- archive (or static).

Ideally the same set of procedures should be used for each type of library.

While Section 6 of the SCMP, Tools, Techniques and Methods', gives background information, this section should describe, in a stepwise manner, how these are applied.

4.5.4.2 SCMP/4.2 Media control

This section should describe the procedure for handling the hardware on which the software resides, such as:

- magnetic disk;
- magnetic tape;

- Read-Only Memory (ROM);
- Erasable Programmable Read-Only Memory (EPROM);
- optical disk.

Whatever media is used, this section should describe the procedures for:

- labelling media (which should be based on ESA PSS-05-0, Part 2, Section 3.2.1);
- storing the media (e.g. fire-proof safes, redundant off-site locations);
- recycling the media (e.g. always use new magnetic tapes when archiving).

4.5.4.3 SCMP/4.3 Change control

This section should define the procedures for processing changes to baselines described in Section 3.2 of the plan.

4.5.4.3.1 SCMP/4.3.1 Levels of authority

This section should define the level of authority required to authorise changes to a baseline (e.g. software librarian, project manager).

4.5.4.3.2 SCMP/4.3.2 Change procedures

This section should define the procedures for processing change proposals to software.

The documentation change process (i.e. URD, SRD, ADD, DDD, SUM) should be based on the procedures defined in ESA PSS-05-0, Part 2, Section 3.2.3.2.1, Documentation Change Procedures'. These procedures use the RID, DCR and DSS forms.

The code change process should be based on the procedures ESA PSS-05-0, Part 2, Section 3.2.3.2.2, Software Problem Reporting procedures'. These procedures use the SPR, SCR and SMR forms.

4.5.4.3.3 SCMP/4.3.3 Review board

This section should define the:

- membership of the review board (e.g. DD/R, SRB);
- levels of authority required for different types of change.

The second point implies that the review board may delegate some of their responsibilities for change.

4.5.4.3.4 SCMP/4.3.4 Interface control

This section should define the procedures for controlling the interfaces. In particular it should define the:

- change control process for ICDs;
- status accounting process for ICDs.

4.5.4.3.5 SCMP/4.3.5 Support software change procedures

This section should define the change control procedures for support software items. Support software items are not produced by the developer, but may be components of the delivered system, or may be tools used to make it. Examples are:

- commercial software;
- reused software;
- compilers;
- linkers.

4.5.5 SCMP/5 CONFIGURATION STATUS ACCOUNTING

This section should:

- define how configuration item status information is to be collected, stored, processed and reported;
- identify the periodic reports to be provided about the status of the CIs, and their distribution;
- state what dynamic inquiry capabilities, if any, are to be provided;
- describe how to implement any special status accounting requirements specified by users.

In summary, this section defines how the project will keep an audit trail of changes.

4.5.6 SCMP/6 TOOLS, TECHNIQUES AND METHODS FOR SCM

This section should describe the tools, techniques and methods to support:

- configuration identification (e.g. controlled allocation of identifiers);
- configuration item storage (e.g. source code control systems);
- configuration change control (e.g. online problem reporting systems);
- configuration status accounting (e.g. tools to generate accounts).

4.5.7 SCMP/7 SUPPLIER CONTROL

This section should describe the requirements for software configuration management to be placed on external organisations such as:

- subcontractors;
- suppliers of commercial software (i.e. vendors).

This section should reference the review and audit procedures in the SVVP for evaluating the acceptability of supplied software.

This section may reference contractual requirements.

4.5.8 SCMP/8 RECORDS COLLECTION AND RETENTION

This section should:

- identify the software configuration management records to be retained (e.g. CIDLs, RIDs, DCRs, DSSs, SPRs, SCRs, SMRs, configuration status accounts);
- state the methods to be used for retention (e.g. fire-proof safe, on paper, magnetic tape);
- state the retention period (e.g. retain all records for all baselines or only the last three baselines).

4.6 EVOLUTION

4.6.1 UR phase

By the end of the UR review, the SR phase section of the SCMP must be produced (SCMP/SR) (SCM42). The SCMP/SR must cover the configuration management procedures for all documentation, CASE tool outputs or prototype code produced in the SR phase (SCM43).

Since the SCMP/SR is the first section to be produced, Section 3.2 of the SCMP should identify the libraries in which the URD, SRD, SCMP and SVVP will be stored. Section 3.3 of the SCMP should describe the UR/R and SR/R board.

4.6.2 SR phase

During the SR phase, the AD phase section of the SCMP must be produced (SCMP/AD) (SCM44). The SCMP/AD must cover the configuration management procedures for documentation, CASE tool outputs or prototype code produced in the AD phase (SCM45). Unless there is a good reason to change (e.g. different CASE tool used), SR phase procedures should be reused.

Section 3.2 should identify the libraries in which the ADD will be stored. Section 3.3 should describe the AD/R board.

4.6.3 AD phase

During the AD phase, the DD phase section of the SCMP must be produced (SCMP/DD) (SCM46). The SCMP/DD must cover the configuration management procedures for documentation, deliverable code, CASE tool outputs or prototype code produced in the DD phase (SCM47). Unless there is a good reason to change, AD phase procedures should be reused.

Section 3.2 should identify the libraries in which the DDD, SUM, and code will be stored. Section 3.3 should describe the DD/R board.

4.6.4 DD phase

During the DD phase, the TR phase section of the SCMP must be produced (SCMP/TR) (SCM48). The SCMP/TR must cover the procedures for the configuration management of the deliverables in the operational environment (SCM49).

This page is intentionally left blank.

APPENDIX A
GLOSSARY

A.1 LIST OF TERMS

Terms used in this document are consistent with ESA PSS-05-0 [Ref 1] and ANSI/IEEE Std 610.12 [Ref 2]. This section defines the software configuration management terms used in this guide.

Archive library

A software library used for storing CIs in releases or retired baselines.

Backup

A copy of a software item for use in the event of the loss of the original.

Baseline

A configuration item that has been formally reviewed and agreed upon, that thereafter serves as a basis for further development, and that can be changed only through formal change control procedures.

Change control

An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification [Ref. 2].

Charge-in

The process of inserting or replacing a configuration item in a master library.

Charge-out

The process of obtaining a copy of a configuration item from a master library and preventing charge-out operations on the item until it is charged back in.

Configuration identification

An element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation [Ref. 2].

Configuration item

A aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process [Ref. 2].

Configuration item identifier

A data structure that describes the name, type and version number of a configuration item.

Configuration item list

A catalogue of configuration items in a baseline or release.

Configuration item storage

The process of storing configuration items in software libraries and on hardware media.

Configuration status accounting

An element of configuration management, consisting of the recording and reporting of information needed to manage a configuration effectively [Ref. 2].

Control authority

The individual or group who decides upon the changes to a configuration item; a configuration item cannot be changed without the control authority's consent.

Controlled configuration item

A configuration item that has been accepted by the project for integration into a baseline. Controlled configuration items are stored in master libraries. Controlled configuration items are subject to formal change control.

Derivation record

A record of one event in the life of the configuration item (e.g. source code change, compilation, testing etc).

Development history

A set of time-ordered derivation records; there is a development history for each configuration item.

Development library

A software library that is used for storing software components that are being coded (or modified) and unit tested.

Document change record

A description of one or more changes to a document.

Document status sheet

An issue-by-issue description of the history of a document.

Formal

Used to describe activities that have explicit and definite rules of procedure (e.g. formal review) or reasoning (e.g. formal method and formal proof).

Issue

A version of a document that has undergone major changes since the previous version.

Level of authority

The position in the project hierarchy that is responsible for deciding upon a change.

Master library

A software library that is used for storing controlled configuration items.

Media control

The process of labelling and storing the hardware items on which the software is stored.

Release

A baseline made available for use.

Review board

The authority responsible for evaluating proposed ... changes, and ensuring implementation of the approved changes; same as configuration control board in Reference 2.

Review item discrepancy

A description of a problem with a document and its recommended solution.

Revision

A version of a document with minor changes from the previous version.

Simultaneous update

The phenomenon of two updates being made by different people on different copies of the same configuration item; one of the updates is always lost.

Software change request

A description of the changes required to documentation and code, the responsible staff, schedule and effort.

Software configuration management

A discipline of applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements [Ref. 2].

Software librarian

A person responsible for establishing, controlling, and maintaining a software library.

Software library

A controlled collection of software and related documentation designed to aid in software development, use, or maintenance [Ref. 2].

Software modification report

A report that describes the implementation of a software change request.

Software problem report

A report that describes a problem with the software, the urgency of the problem, the environment in which it occurs, and the recommended solution.

Software release note

A document that describes the changes and configuration items in a software release

Software review board

The name for the review board in the OM phase.

Variant

A configuration item that meets special requirements.

A.2 LIST OF ACRONYMS

AD	Architectural Design
AD/R	Architectural Design Review
ADD	Architectural Design Document
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
AT	Acceptance Test
BSSC	Board for Software Standardisation and Control
CASE	Computer Aided Software Engineering
DBMS	Data Base Management System
DCR	Document Change Record
DD	Detailed Design and production
DD/R	Detailed Design and production Review
DDD	Detailed Design and production Document
DSS	Document Status Sheet
EPROM	Erasable Programmable Read-Only Memory
ESA	European Space Agency
ICD	Interface Control Document
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organisation
IT	Integration Test
PA	Product Assurance
PCTE	Portable Common Tools Environment
PSS	Procedures, Specifications and Standards
QA	Quality Assurance
RID	Review Item Discrepancy
ROM	Read-Only Memory
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	Software Change Request
SMR	Software Modification Report
SPM	Software Project Management
SPMP	Software Project Management Plan
SPR	Software Problem Report
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SR	Software Requirements
SR/R	Software Requirements Review
SRD	Software Requirements Document
SRN	Software Release Note

ST	System Test
STD	Software Transfer Document
SUM	Software User Manual
SVVP	Software Verification and Validation Plan
UR	User Requirements
UR/R	User Requirements Review
URD	User Requirements Document
UT	Unit Test

This page is intentionally left blank.

APPENDIX B
REFERENCES

1. ESA Software Engineering Standards, ESA PSS-05-0 Issue 2 February 1991.
2. IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Std 610.12-1990.
3. IEEE Standard for Software Configuration Management Plans, ANSI/IEEE Std 828-1990.
4. IEEE Guide to Software Configuration Management, ANSI/IEEE Std 1042-1987.
5. The STARTs Guide - a guide to methods and software tools for the construction of large real-time systems, NCC Publications, 1987.
6. Managing the Software Process, Watts S. Humphrey, SEI Series in Software Engineering, Addison-Wesley, August 1990.
7. Portable Common Tool Environment (PCTE) Abstract Specification, Standard ECMA-149, European Computer Manufacturers Association, December 1990.
8. Requirements for Ada Programming Support Environments, STONEMAN, U.S Department of Defense, February 1980.

This page is intentionally left blank

APPENDIX C
MANDATORY PRACTICES

This appendix is repeated from ESA PSS-05-0, appendix D.9.

SCM01 All software items, for example documentation, source code, object or relocatable code, executable code, files, tools, test software and data, shall be subjected to configuration management procedures.

SCM02 The configuration management procedures shall establish methods for identifying, storing and changing software items through development, integration and transfer.

SCM03 A common set of configuration management procedures shall be used.

Every configuration item shall have an identifier that distinguishes it from other items with different:

SCM04 • requirements, especially functionality and interfaces;

SCM05 • implementation.

SCM06 Each component defined in the design process shall be designated as a CI and include an identifier.

SCM07 The identifier shall include a number or a name related to the purpose of the CI.

SCM08 The identifier shall include an indication of the type of processing the CI is intended for (e.g. filetype information).

SCM09 The identifier of a CI shall include a version number.

SCM10 The identifier of documents shall include an issue number and a revision number.

SCM11 The configuration identification method shall be capable of accommodating new CIs, without requiring the modification of the identifiers of any existing CIs.

SCM12 In the TR phase, a list of configuration items in the first release shall be included in the STD.

SCM13 In the OM phase, a list of changed configuration items shall be included in each Software Release Note (SRN).

SCM14 An SRN shall accompany each release made in the OM phase.

As part of the configuration identification method, a software module shall have a standard header that includes:

SCM15 • configuration item identifier (name, type, version);

SCM16 • original author;

SCM17 • creation date;

SCM18 • change history (version/date/author/description).

All documentation and storage media shall be clearly labelled in a standard format, with at least the following data:

SCM19 • project name;

SCM20 • configuration item identifier (name, type, version);

SCM21 • date;

SCM22 • content description.

To ensure security and control of the software, at a minimum, the following software libraries shall be implemented for storing all the deliverable components (e.g. documentation, source and executable code, test files, command procedures):

SCM23 • Development (or Dynamic) library;

SCM24 • Master (or Controlled) library;

SCM25 • Static (or Archive) library.

SCM26 • Static libraries shall not be modified.

SCM27 Up-to-date security copies of master and static libraries shall always be available.

SCM28 Procedures for the regular backup of development libraries shall be established.

SCM29 The change procedure described (in Part 2, Section 3.2.3.2.1) shall be observed when changes are needed to a delivered document.

SCM30 Software problems and change proposals shall be handled by the procedure described (in Part 2, Section 3.2.3.2.2).

SCM31 The status of all configuration items shall be recorded.

To perform software status accounting, each software project shall record:

SCM32 • the date and version/issue of each baseline;

SCM33 • the date and status of each RID and DCR;

SCM34 • the date and status of each SPR, SCR and SMR;

SCM35 • a summary description of each Configuration Item.

SCM36 As a minimum, the SRN shall record the faults that have been repaired and the new requirements that have been incorporated.

SCM37 For each release, documentation and code shall be consistent.

SCM38 Old releases shall be retained, for reference.

SCM39 Modified software shall be retested before release.

SCM40 All software configuration management activities shall be documented in the Software Configuration Management Plan (SCMP).

SCM41 Configuration management procedures shall be in place before the production of software (code and documentation) starts.

SCM42 By the end of the UR review, the SR phase section of the SCMP shall be produced (SCMP/SR).

SCM43 The SCMP/SR shall cover the configuration management procedures for documentation, and any CASE tool outputs or prototype code, to be produced in the SR phase.

SCM44 During the SR phase, the AD phase section of the SCMP shall be produced (SCMP/AD).

SCM45 The SCMP/AD shall cover the configuration management procedures for documentation, and CASE tool outputs or prototype code, to be produced in the AD phase.

SCM46 During the AD phase, the DD phase section of the SCMP shall be produced (SCMP/DD).

SCM47 The SCMP/DD shall cover the configuration management procedures for documentation, deliverable code, and any CASE tool outputs or prototype code, to be produced in the DD phase.

SCM48 During the DD phase, the TR phase section of the SCMP shall be produced (SCMP/TR).

SCM49 The SCMP/TR shall cover the procedures for the configuration management of the deliverables in the operational environment.

APPENDIX D

FORM TEMPLATES

Template forms are provided for:

DCR	Document Change Record
DSS	Document Status Sheet
RID	Review Item Discrepancy
SCR	Software Change Request
SMR	Software Modification Report
SPR	Software Problem Report
SRN	Software Release Note

DOCUMENT CHANGE RECORD	DCR NO	
	DATE	
	ORIGINATOR	
	APPROVED BY	
1. DOCUMENT TITLE:		
2. DOCUMENT REFERENCE NUMBER:		
3. DOCUMENT ISSUE/REVISION NUMBER:		

4. PAGE	5. PARAGRAPH	6. REASON FOR CHANGE

DOCUMENT STATUS SHEET			
1. DOCUMENT TITLE:			
2. DOCUMENT REFERENCE NUMBER:			
3. ISSUE	4. REVISION	5. DATE	6. REASON FOR CHANGE

REVIEW ITEM DISCREPANCY	RID NO	
	DATE	
	ORIGINATOR	
1. DOCUMENT TITLE:		
2. DOCUMENT REFERENCE NUMBER:		
3. DOCUMENT ISSUE/REVISION NUMBER:		
4. PROBLEM LOCATION:		
5. PROBLEM DESCRIPTION:		
6. RECOMMENDED SOLUTION;		
7. AUTHOR S RESPONSE:		
8. REVIEW DECISION: CLOSE/UPDATE/ACTION/REJECT (underline choice)		

SOFTWARE CHANGE REQUEST	SCR NO	
	DATE	
	ORIGINATOR	
	RELATED SPRs	
1. SOFTWARE ITEM TITLE:		
2. SOFTWARE ITEM VERSION/RELEASE NUMBER:		
3. PRIORITY: CRITICAL/URGENT/ROUTINE (underline choice)		
4. CHANGES REQUIRED:		
5. RESPONSIBLE STAFF:		
6. ESTIMATED START DATE, END DATE AND MANPOWER EFFORT		
7. ATTACHMENTS:		
8. REVIEW DECISION: CLOSE/UPDATE/ACTION/REJECT (underline choice)		

SOFTWARE MODIFICATION REPORT	SMR NO	
	DATE	
	ORIGINATOR	
	RELATED SCRs	
1. SOFTWARE ITEM TITLE:		
2. SOFTWARE ITEM VERSION/RELEASE NUMBER:		
3. CHANGES IMPLEMENTED:		
4. ACTUAL START DATE, END DATE AND MANPOWER EFFORT:		
5. ATTACHMENTS		(tick as appropriate)

SOFTWARE PROBLEM REPORT	SPR NO	
	DATE	
	ORIGINATOR	
1. SOFTWARE ITEM TITLE:		
2. SOFTWARE ITEM VERSION/RELEASE NUMBER:		
3. PRIORITY: CRITICAL/URGENT/ROUTINE (underline choice)		
4. PROBLEM DESCRIPTION:		
5. DESCRIPTION OF ENVIRONMENT:		
6. RECOMMENDED SOLUTION;		
7. REVIEW DECISION: CLOSE/UPDATE/ACTION/REJECT (underline choice)		
8. ATTACHMENTS:		

SOFTWARE RELEASE NOTE	SRN NO	
	DATE	
	ORIGINATOR	
1. SOFTWARE ITEM TITLE:		
2. SOFTWARE ITEM VERSION/RELEASE NUMBER:		
3. CHANGES IN THIS RELEASE:		
4. CONFIGURATION ITEMS INCLUDED IN THIS RELEASE:		
5. INSTALLATION INSTRUCTIONS:		

APPENDIX D INDEX