

CC.A Guidelines on What Configuration Items to include for Software Change Control

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 (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 project plan.

Configuration 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 might rely on the file management facilities of the operating system. It is no accident therefore that CIs are often discrete files.

CC.A.1 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 (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.

CC.A.2 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 minimized. 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.

CC.B Guidelines on How to identify CIs, their Control Authority and provide for History Descriptions

Each configuration item must possess an identifier. The identifier contains the name, type and version attributes of the configuration item. The change control 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 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 change control 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. 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.

CC.B.1 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), the control authority changes to match the higher verification status of the CI.

CC.B.1.1 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.

CC.B.1.2 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 system libraries. On most projects the software project manager is supported by a system librarian.

CC.B.1.3 Refer also Section 5.2 of Software Configuration & Management Process Guide for Application Software available in ITSD.

CC.B.2 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 the Document Change Records (DCRs). For source code, the development history is stored in the module header. Change records in module headers should reference the Software Change Request 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 summarize the changes to CIs that have been revised in response to change request.

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).

CC.C Documentation Template for the Software Change Control Plan (CCP)

CC.C.1 Introduction

The following subsections should provide an introduction to the plan.

CC.C.1.1 Purpose

This section should:

- briefly define the purpose of the particular CCP
- specify the intended readership of the CCP

CC.C.1.2 Scope

This section should identify the:

- configuration items to be managed;
- configuration management activities in this plan;
- organizations the plan applies to;
- phase of the life cycle the plan applies to.

CC.C.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.

CC.C.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 organization should be included.

CC.C.2 This section should describe the organization of change control management, and the associated responsibilities. It should define the roles to be carried out.

CC.C.2.1 Organization

This section should:

- identify the organizational roles that influence the software change control function (e.g. project managers, programmers, quality assurance personnel and review boards);
- describe the relationships between the organizational roles;
- describe the interface with the user organization.

Relationships between the organizational roles may be shown by means of an organization chart.

CC.C.2.2 Responsibilities

This section should identify the:

- software change control functions each organizational role is responsible for (e.g. identification, storage, change control, status accounting);
- responsibilities of each organizational role in the review, audit and approval process;
- responsibilities of the users in the review, audit and approval process.

CC.C.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 organizations responsible for the systems or subsystems with which the software interfaces;
- points of contact in the external organizations for jointly managing the interface;

- groups responsible for the management of each interface.

CC.C.2.4 Change Control Plan implementation

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

- readiness of the change control system for use;
- establishment of the Software Review Group (SRG);
- establishment of baselines;
- release of products.

The scheduling of the change control management resources should be shown (e.g. availability of software librarian, software change control tools and SRG). This section may cross-reference the software project plan.

CC.C.2.5 Applicable policies, directives and procedures

This section should:

- identify all applicable software change control policies, directives or procedures to be implemented as part of this plan (corporate software change control documents may be referenced here, with notes describing the parts of the documents that apply);
- describe any software change control, policies, directives or procedures specific to this project, for example:
 - project-specific interpretations of corporate software change control documents;
 - level of authority required for each level of control;
 - level of review, testing or assurance required for promotion.

CC.C.3 Configuration Identification

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

CC.C.3.1 Conventions

This section should:

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

CC.C.3.2 Baselines

For each baseline, this section should give the:

- identifier of baseline;
- contents, i.e. the:
 - software itself (e.g. User Requirements Document, Software Requirements Document, Design Specifications, modules, executables, Software User Manual, Software Project Plan).
 - tools for making derived items in the baseline (e.g. compiler, linker and build procedures);
 - test software (e.g. data, harnesses);
 - SPRs etc that relate to the baseline;
- review and approval events, and the acceptance criteria, associated with establishing each baseline;
- participation required of developers and users in establishing baselines.

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. 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 software project plan, and, for code, to the software components described in Software Requirement Document and Design Specifications.

CC.C.4 Configuration Control

Configuration control covers only configuration item change control.

CC.C.4.1 Code (and Document) Control

This section should describe the software library handling procedures, including

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

This section should describe, in a stepwise manner, how these are applied.

CC.C.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;
- storing the media (e.g. fire-proof safes, redundant off-site locations);
- recycling the media (e.g. always use new magnetic tapes when archiving).

CC.C.4.3 Change Control

This section should define the procedures for processing changes to baselines described above.

a) Levels of authority

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

b) Change procedures

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

c) Review group

This section should define the:

- membership of the review group
- levels of authority required for different types of change.

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

d) Interface control

This section should define the procedures for controlling the interfaces.

e) 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.

CC.C.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.

CC.C.6 Tools, Techniques and Methods for Change Control

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), elaborate on the structure and access rights of the central library;
- configuration change control (e.g. online problem reporting systems);
- configuration status accounting (e.g. tools to generate accounts).

CC.C.7 Supplier Control

This section should describe the requirements for software change control to be placed on external organizations such as:

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

CC.C.8 Records Collection and Retention

This section should:

- identify the software change control records to be retained (e.g. 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).

CC.D Discussion of Software Change Control Tools in terms of their Capabilities

CC.D.1 Introduction

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

This chapter does not describe particular tools. Instead, this chapter discusses software change control 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.

There are basically four types of toolset:

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

This classification is used to organize software change control tools. Each toolset includes the capabilities of preceding toolsets.

CC.D.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.

CC.D.2.1 A basic toolset should permit the structured definition, identification and storage of configuration items.

CC.D.2.2 The file system 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.

CC.D.2.3 A basic toolset should permit the structured storage of configuration items.

CC.D.2.4 The file systems of operating systems should allow the creation of directory trees for storing files.

CC.D.2.5 A basic toolset should provide a librarian system that supports:

- a) insertion of modules;
- b) extraction of modules;
- c) replacement of modules by updated modules;
- d) deletion of modules;
- e) production of cross-reference listings to show which modules refer to which;
- f) storage of the transaction history;
- g) all the above features for both source (i.e. text) and object (i.e. binary) files.

CC.D.2.6 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.

CC.D.2.7 A basic toolset should provide security features so that access to CIs can be restricted to authorized personnel.

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

CC.D.2.8 A basic toolset should provide facilities for comparing source modules so that changes can be identified.

Most operating systems provide a tool for differentiating ASCII files.

CC.D.2.9 A basic toolset should include tools for the systematic backup of CIs. Specifically:

- a) automatic electronic labelling of media;
- b) complete system backup;
- c) incremental backup;
- d) restore;
- e) production of backup logs

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

CC.D.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.

CC.D.3.1 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.

CC.D.3.2 An advanced toolset should minimize the storage space needed.

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

CC.D.3.3 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.

CC.D.3.4 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 operation.

CC.D.3.5 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.

CC.D.3.6 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.

CC.D.3.7 An advanced toolset should provide facilities for the handling of configuration status accounts, specifically:

- a) insertion of new SPR, SCR and SMR status records;
- b) modification of SPR, SCR and SMR status records;
- c) deletion of SPR, SCR and SMR status records;
- d) search of SPR, SCR and SMR records on any field;
- e) differentiation of configuration status accounts, so that changes can be identified;
- f) report generation.

CC.D.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 on-line toolset are listed below.

CC.D.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.

CC.D.5.1 An integrated toolset should recognize 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.

CC.D.5.2 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).

CC.E Form Template for preparing the Software Problem Report (SPR)

Originator:	SPR No.:
	Date:
1. Software Item Title:	
2. Software Item Version/Release No.:	
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. Review Decision: Close/Update/Action/Reject (underline choice)	

CC.F Form Template for preparing the Software Change Report (SCR)**Software Change Request**

Originator:	SRC No.:
Related SPRs:	Date:
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 Report:	
7. Attachments:	

CC.G Form Template for documenting Software Modification Report (SMR)

Originator:	SMR No.:										
Related SCRs:	Date:										
1. Software Item Title:											
2. Software Item Version/Release Number:											
3. Changes Implemented:											
4. Actual Start Date, End Date and Manpower Effort:											
5. Attachments	<table> <tr> <td>Source Code</td> <td><input type="checkbox"/></td> </tr> <tr> <td>Test Procedures</td> <td><input type="checkbox"/></td> </tr> <tr> <td>Test Data</td> <td><input type="checkbox"/></td> </tr> <tr> <td>Test Results</td> <td><input type="checkbox"/></td> </tr> <tr> <td>Documentation Updates</td> <td><input type="checkbox"/></td> </tr> </table>	Source Code	<input type="checkbox"/>	Test Procedures	<input type="checkbox"/>	Test Data	<input type="checkbox"/>	Test Results	<input type="checkbox"/>	Documentation Updates	<input type="checkbox"/>
Source Code	<input type="checkbox"/>										
Test Procedures	<input type="checkbox"/>										
Test Data	<input type="checkbox"/>										
Test Results	<input type="checkbox"/>										
Documentation Updates	<input type="checkbox"/>										

CC.H Form Template for documenting a Software Release Note (SRN)

Originator:	SRN No.:
Approved by:	Date:
1. Software Item Title:	
2. Software Item Version/Release No.:	
3. Changes in this Release:	
4. Configuration Items included in this Release:	
5. Installation Instructions:	