

UNIT 7

Software Configuration Management

(7 Hours) — Software Project Management (CACS407)

NOTE: These notes may contain errors. Please verify important points, definitions and calculations with your textbook and teacher before relying on them.

7.1 Concept of Software Configuration Management

During development a software project produces many items (source code, documents, test cases, data, tools) and these change continuously. **Software Configuration Management (SCM)** is the discipline of identifying, organising and controlling these items and their changes so that the integrity and traceability of the product are maintained throughout its life.

Term	Meaning
Configuration	The set of all items (and their versions) that make up a product at a given time.
Software Configuration Item (SCI)	Any work product placed under configuration control: source code, SRS, design document, test plan, user manual, build scripts.
Version / revision	A particular state of an item at a point in time.
Release	A version delivered to a customer or users.
Baseline	An approved, frozen version of one or more items that can be changed only through formal change control.

A simple way to remember it: SCM answers the questions *What do we have? Who changed what, when and why? Which version is in production? Can we get back to an earlier version?*

7.2 Requirement for SCM

7.2.1 Why SCM is needed

- Many people work on the same files at the same time; changes may overwrite each other.
- Requirements and designs change; uncontrolled change causes chaos (scope creep, 'it worked yesterday').
- Several versions (customer releases, test builds, bug fixes) must be maintained in parallel.
- The team needs to reproduce any earlier release and find out which change introduced a defect.

- Quality standards (ISO 9001, CMMI) and contracts demand traceable, auditable change control.
- Reduces rework and integration problems, and helps in recovery from loss or errors.

7.2.2 Typical problems without SCM

Problem	Description
Double maintenance	The same fix is made in several copies of code.
Shared data	Two developers edit the same file; one change is lost.
Simultaneous update	No record of who changed which part.
Lost versions	Cannot rebuild the exact released version.
Unauthorised change	Untested or unapproved modifications reach the product.

7.3 Elements of SCM

SCM is made up of the following elements (activities):

Element	Description
Component (configuration) identification	Choosing the items to control, naming them uniquely and defining their relationships and versions.
Version control	Managing the different versions of the items and enabling parallel development (branching, merging).
Change control	Formal procedure to request, evaluate, approve, implement and verify changes.
Configuration building / release management	Assembling the right versions into a build and delivering releases.
Configuration audit	Checking that the product and its records are consistent with the requirements and approved changes.
Configuration status accounting (reporting)	Recording and reporting the status of items and changes.

7.4 Baseline

A **baseline** is a specification or product that has been formally reviewed and agreed, and thereafter serves as the basis for further development; it can be changed only through formal change control. (IEEE Std 610.12.)

- **Functional baseline** — approved system requirements.
- **Allocated baseline** — approved software requirements specification (SRS).
- **Design baseline** — approved design documents.
- **Product baseline** — tested code and documentation ready for release.
- **Operational baseline** — the product in operation (maintenance).

Before an item becomes a baseline, it can be changed informally by its author. After baselining, a change requires formal approval. Baselines are named or tagged, so any of them can be recovered later.

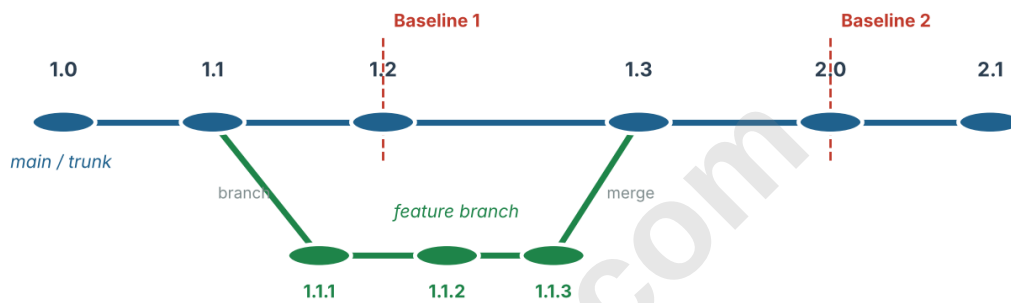


Figure 7.1 — Versions, branches, merges and baselines

7.5 SCM Repository

The **SCM repository** is the central database (or distributed set of databases) that stores all configuration items together with their complete history.

7.5.1 Features of a good repository

- Stores every version of every item and the **metadata** (author, date, comment, change request number).
- **Access control** and security: who may read, write or approve.
- **Locking or merging** to handle simultaneous changes.
- Support for **branching, merging and tagging/labelling** of baselines.
- **Traceability**: links between requirements, design, code, tests and change requests.
- **Reporting** and querying; integrity checks and backup.
- **Integration** with build, issue-tracking and testing tools.

7.5.2 Check-out and check-in

- **Check-out:** a developer copies a version from the repository to a private workspace to modify it.
- **Check-in (commit):** the modified item is returned and stored as a new version with a comment.
- Two models of concurrency: **lock–modify–unlock** (only one editor at a time) and **copy–modify–merge** (many edit, then merge).

7.6 Versioning and Version Control

Version control (revision control) is the management of changes to documents and source code so that previous versions can be recovered and different lines of development can proceed in parallel.

7.6.1 Version numbering

- **Sequential revision numbers** such as 1.1, 1.2, 1.3 (CVS, per file) or 1, 2, 3 for the whole repository (SVN).
- **Release numbering** such as *major.minor.patch* (for example 2.4.1): major = incompatible/big change, minor = new features, patch = bug fix.
- **Hash identifiers** (Git) that identify each commit uniquely by content.
- **Tags/labels** give meaningful names (such as 'release-2.0') to a set of versions.

7.6.2 Key concepts

Concept	Meaning
Trunk / main	The main line of development.
Branch	A separate line of development (feature, release, bug fix).
Merge	Combining changes from one branch into another.
Conflict	Same part of a file changed in two ways; must be resolved manually.
Tag	A name attached to a particular state of the repository (e.g. a release).
Diff	Difference between two versions.
Revert / rollback	Return to an earlier version.

7.6.3 Types of version control systems

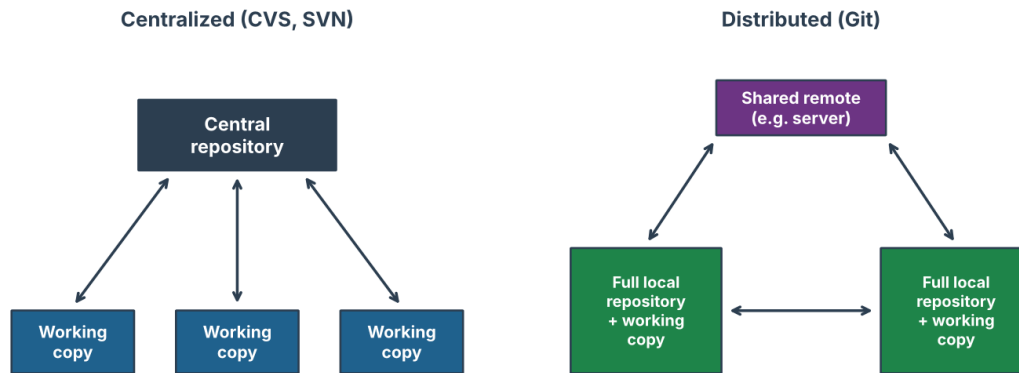


Figure 7.2 — Centralized and distributed version control

Type	Description	Examples
Local	Revisions kept in a database on one computer.	RCS, SCCS
Centralized (client–server)	One central repository; users keep a working copy only.	CVS, Subversion
Distributed	Every user has a full copy of the repository, including history.	Git, Mercurial

7.7 SCM Process

The SCM process covers all SCM activities in the life cycle. A typical sequence:

- **Plan SCM:** write the SCM plan (items, naming, roles, tools, baselines, procedures).
- **Identify** configuration items and naming conventions.
- **Control** versions through the repository; baseline at milestones.
- **Control changes** through the change control process.
- **Build and release** from baselined items.
- **Audit** configurations and **report status** regularly.

Note: Roles: Configuration Manager (runs the SCM process), Change Control Board (CCB, approves changes), developers, testers/QA, and release manager.

7.8 Change Control Process

Change is inevitable, but it must be controlled. The **change control process** ensures that each change is requested, evaluated, approved, implemented and verified in an orderly way.

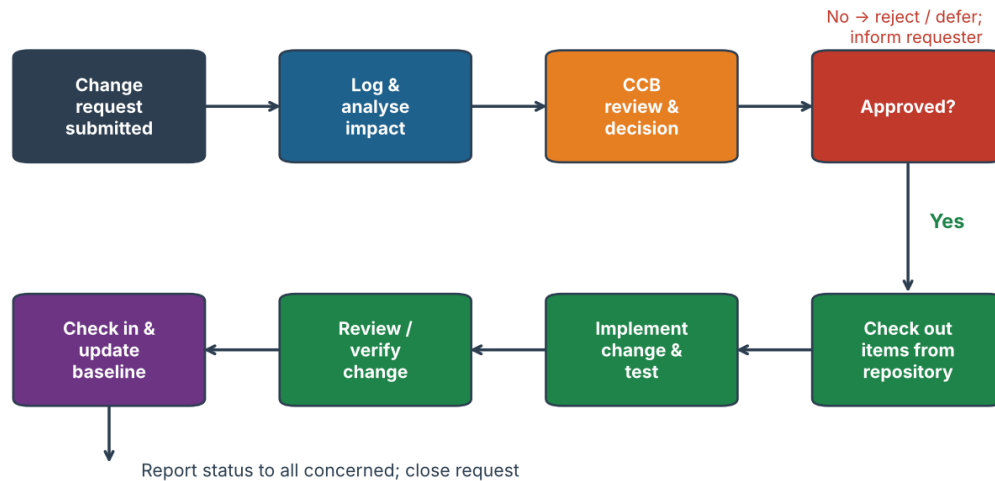


Figure 7.3 — Typical change control process

Step	Activity
1. Change request (CR)	A user, tester or developer fills in a Change Request form: description, reason, priority, originator.
2. Evaluation / impact analysis	Technical effort, cost, schedule, risk and affected items are analysed.
3. Decision by CCB	The Change Control Board approves, rejects or defers the change and sets priority.
4. Implementation	Items are checked out, modified in a separate workspace/branch, and tested.
5. Verification / review	Reviews, regression testing and QA confirm the change works and nothing else broke.
6. Check-in & release	Items are checked in, a new version/baseline is created and the change is released.
7. Reporting & closure	Status is reported to everyone concerned; the CR is closed and recorded.

- Emergency changes follow a fast-track procedure but are still recorded and reviewed afterwards.
- A common contents list for a CR form: ID, date, requester, description, affected items, priority, estimated effort, status, approver's signature.

7.9 Configuration Audit and Status Reporting

7.9.1 Configuration audit

A configuration audit checks that the items in a baseline match the approved requirements and that all approved changes have been made and no unapproved changes exist.

Audit type	Question answered
Functional configuration audit (FCA)	Does the product perform as required and does it meet the specified functionality?
Physical configuration audit (PCA)	Does the delivered product (code, documentation, media) match what was documented and approved?
Process audit	Were the SCM procedures followed?

- Typical checks: Were all approved changes implemented? Were reviews done? Were the standards followed? Are the right versions in the release? Are the records complete?
- Audits are carried out before releases and at milestone baselines, usually by QA or an independent team.

7.9.2 Configuration status accounting (reporting)

Status reporting records and reports information needed to manage the configuration: what is the status of each item and change request, who made changes and when, which baseline contains what.

- List of baselines and their contents; version history of each item.
- Status of change requests (submitted, approved, in progress, tested, closed).
- Number of changes per item/module (identifies unstable parts), open vs closed change requests.
- Differences between two builds or releases (release notes).

7.10 Case Study: Version Control Tools (Git, CVS, SVN)

7.10.1 CVS (Concurrent Versions System)

- Early open-source centralized system (1980s–1990s), built on the older RCS; widely used for open-source projects before 2000.
- Versions each **file** separately (revision such as 1.4) and uses **copy–modify–merge**.
- Weaknesses: commits are **not atomic** (a failed commit can leave a partial update); renaming/moving files and directories is poorly supported; limited binary-file support; branching and merging are cumbersome.
- Typical commands: **cv**s checkout, **cv**s update, **cv**s add, **cv**s commit, **cv**s tag, **cv**s log, **cv**s diff.

7.10.2 SVN (Apache Subversion)

- Created in 2000 (CollabNet) to be 'a better CVS'; centralized client–server system, now an Apache project.
- **Atomic commits**: a commit either fully succeeds or fails.
- A single **repository-wide revision number** increases with each commit.
- Versions directories, renames and file metadata; handles binary files better.
- Branches and tags are cheap copies (*svn copy*), conventionally in **trunk / branches / tags** directories.
- Typical commands: **svn checkout, svn update, svn add, svn commit, svn diff, svn log, svn copy, svn merge.**

7.10.3 Git

- Created in 2005 by Linus Torvalds for Linux kernel development; **distributed**.
- Each clone has the **complete history**, so most operations (commit, log, branch) are local, fast and work offline.
- Data is stored as snapshots identified by **SHA hashes**; integrity is checked automatically.
- **Branching and merging are lightweight**, supporting workflows such as feature branches and pull requests.
- Includes a **staging area** (index) between the working directory and the repository.
- Hosting services: GitHub, GitLab, Bitbucket.
- Typical commands: **git init, git clone, git add, git commit, git status, git log, git branch, git switch/checkout, git merge, git pull, git push, git tag, git diff.**

Example 7.1 — A simple Git workflow

1. `git clone <url>` — copy the repository.
2. `git switch -c login-feature` — create and switch to a new branch.
3. Edit files; `git add .` then `git commit -m "Add login form"`.
4. `git switch main`; `git merge login-feature` — merge the finished feature.
5. `git tag v1.0` — mark a release/baseline; `git push --tags` — share it.

7.10.4 Comparison

Feature	CVS	SVN	Git
Model	Centralized	Centralized	Distributed
Unit of versioning	Individual files	Whole tree (directories, files)	Snapshots of whole repository
Revision identifier	Per-file numbers (1.1, 1.2)	Global revision number	Commit hash
Atomic commit	No	Yes	Yes
Offline work	Very limited	Limited (local copy only)	Full (local history)
Branching / merging	Difficult	Easier, but merging can be tricky	Fast and lightweight; very strong merge tracking
Rename / move tracking	Poor	Supported	Detected through content
Current use	Mostly legacy	Still used in some organisations	Most widely used today

Note: Choosing a tool: small teams with simple needs may use any; for distributed teams, many branches and open-source style collaboration, Git is usually preferred. Always combine the tool with a clear branching strategy and change control process.

Key Takeaways

- ✓ SCM controls changes to configuration items (code, documents, tests) to maintain integrity and traceability.
- ✓ Elements: identification, version control, change control, build/release, audit, status reporting.
- ✓ A baseline is an approved version that can change only through formal change control.
- ✓ The repository stores all versions and metadata; users check out, modify and check in (or commit).
- ✓ Change control: request → impact analysis → CCB decision → implement → verify → release → report.
- ✓ Audits (functional and physical) verify the product; status reports show state of items and changes.
- ✓ CVS and SVN are centralized; Git is distributed with full local history and cheap branching.

— End of Unit 7 —