

General principles for version and configuration control

- Configuration management (CM)
 - **A discipline for managing the development of complex systems**
- Software configuration management (SCM)
 - **A discipline for managing the development of software systems**
- Special properties of software configurations and systems:
 - **Software is changed much more often than hardware**
 - **Configuration management for software can potentially be given more computer support, since the system components are stored in machine-processable form**

Five functions for a configuration management system:

- Identification
 - **Reliable identification/naming of modules and configurations**
- Change tracking
 - **To keep track of changes: Who, When and Why?**
- Version selection and base configurations
 - **To put the right versions of modules together into a software application, when can be tested; or selecting a base configuration**
- Software manufacture
 - **Automating the integration and manufacturing process**
 - **Can contain steps such as: preprocessing, compiling, linking, regression testing and formatting**
- Managing simultaneous update
 - **Several programmers can change the same module at the same time**

Questions about identification

Reliable identification/name of single modules and configurations

- "This program worked yesterday. What happened?"
- "I can't reproduce the error in this configuration."
- "I fixed this problem long ago. Why did it reappear?"
- "The online documentation doesn't match the program."
- "Do we have the latest version?"

Questions about managing change

To keep track of changes: Who did it, When did he do it, and Why did he do it?

- "Has this problem been fixed?"
- "Which bug fixes went into this copy?"
- "This seems like an obvious change. Was it tried before?"
- "Who is responsible for this modification?"
- "Were these independent changes merged?"

Version selection and base configurations (baselining)

To put together and select the right version of the software components (i.e. modules) for testing or for creating a base configuration

- "How do I configure a test system that contains my temporary fixes to the last baseline, and the released fixes of all other components?"
- "Given a list of fixes and enhancements, how do I configure a system that incorporates them?"
- "This enhancement won't be ready until the next release. How do I configure it out of the current baseline?"
- "How exactly does this version differ from the Baseline?"

Software manufacture

Automate the integration and manufacture process

This can contain steps such as: preprocessing, compilation, linking, regression testing and formatting

- "I just fixed that. Was something not recompiled?"
- "How much recompilation will this change cost?"
- "Did we deliver an up-to-date binary version to the customer?"
- "I wonder whether we applied the processing steps in the right order?"
- "How exactly was this configuration produced?"
- "Were all regression tests performed on this version?"

Managing simultaneous update

Several programmers are modifying the same module simultaneously

- "Why did my change in this module disappear?"
- "What happened to my unfinished modules while I was out of town?"
- "How do I merge these changes into my version?"
- "Do our changes conflict?"

Basic SCM (Software Configuration Control) concepts

- Software object
 - Any kind of identifiable, machine readable document generated during the project
 - Example: requirement spec, design spec, program source code, test programs, test data, etc.
 - Unique identifier, object contents, attributes
 - The object content is IMMUTABLE. Any change creates a new object
- Source code object
 - Is created manually, e.g. using an editor
- Derived object
 - Is created automatically by some program, without manual interaction, using some other software object as input
 - Example: program generation, compilation
 - Derived objects can be deleted, if they can easily be re-generated

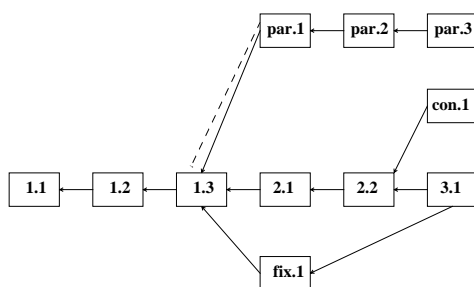
Structure of software objects

- Atomic object
 - An object without any apparent inner structure accessible to the SCM system
 - Generic operations on atomic objects: copy, change names, delete, edit
- Configurations, two types: composite objects and object sequences
- Composite objects
 - Record structure, with fields denoting different types
 - Every field contains an object identifier
- Object Sequence
 - A sequence of object identifiers

Two kinds of versions: revisions, variants

- Revision
 - X is a revision-of Y, if X has been created through a change of Y, and both are software objects
- Variant
 - X and Y are variants if they are identical given a certain abstraction
 - Common case: they have identical functional interfaces
 - Example: Quicksort, Mergesort; (both sorts) VAX-version, SUN-version
- Version groups:
 - A set of objects which are connected by arcs of type *revision-of*, *variant-of*

Example of a version group



← **revision-of**

----- **variant-of**

con.1 conflict at change
par.1 parallel version 1
fix.1 bug fix 1

Operations of version groups

- Check out
- Check in
- Merge

Version groups are usually implemented through delta handlers (e.g. Unix SCCS - Source Code Control System) which stores differences between revisions

SCCS (Source Code Control System)

Unix command: sccs

RCS (Reverse Source code control System)

On the VAX: CMS (Code Management System)

CVS is a more integrated configuration management tool which uses RCS for file versioning. CVS can handle whole projects/directory structures, not just single files.

Generation of configurations

- AND/OR graph describes different possibilities
 - **AND:** which components *have to be* included
 - **OR:** any alternative is possible, e.g. Quicksort/Mergesort, or VAX/SUN
- A base configuration (executable?) has no OR-branches
- Private baseline (base configuration)
 - **Is created by each programmer when they build the system**
- Public baseline (base configuration)
 - **Is create for general use within the project group**