Software Configuration Management and CVS

Basic idea of configuration management (CM) in the software development process is to guarantee the integrity (completeness and intactness) of the software product at any moment of the development.

Aims of CM:

- Structure and discipline in the development process
- reusability of software

Result of CM is

- better software quality and
- increased efficiency of the software development team.

CM is a management task that includes

- People responsible for it
- CM strategy/methods, plans
- Support tools

Aspects of configuration management

Identification. The imposed structure of the product provides access to parts of the product.

Control. Product changes are authorized by a formal procedure that distinguishes different *releases* of the same product and its parts. Consistent releases constitute a *baseline.*

Documentation. Product status (its releases and baselines) are to be documented.

Verification. guarantees completeness and consistency of the components of the product.

Construction Construction of the product from its constituent parts.

Process management CM must support the software life cycle.

Teamwork. CM must allow for teamwork: several teams/developers develop one product.

CM informally

In every product that is to be implemented by a team potential confusion may arise. CM is meant to decrease the confusion. CM must

- identify,
- organize and
- control

changes performed on one product by different developers. The aim of CM is to increase quality by avoiding errors.

Architecture of a CM system

Repository to provide consistency, releases and baselines;

Workareas to allow for parallel development/test and parallel (sic!) changes of the same parts of the software;

Makefiles to support for construction and dependency checks.

Tasks of a CM system

Typical CM activities from the viewpoint of a developer

Check out current product parts

Construct the product from checked out parts (if available)

Check in modifications

Compare own version to one in the repository

Update to actual status

Change the structure of the product: add or remove parts.

Tool support: CVS

Why CVS?

- Allows for parallel development
- Supports remote development
- It is a modular extension of `rcs`
- It's free (GNU).

CVS conflict resolution policy

- Developer copies an object from repository without "locking" it
- Developer modifies the copy and merges the modifications with the original

This allows for *parallel development.*

A *conflict* arises when different developers try to merge modifications of the same source. Here every developer has to merge changes made in the repository before it is allowed to add its modifications to the repository.

This guarantees *completeness and consistency.*

CVS Commands

`cvs command [options] [files...]`

`cvs checkout <module>` Checks out the most actual sources of `<module>` into the current directory. Use this command to create a work-area (WA).

`cvs commit [file ...]` Checks WA files into the repository or fails if the WA is out of date. `commit` asks for a *comment* while checking in: this supports generation of list of modifications (maintenance!)

`cvs diff [file ...]` Runs `diff` between files of WA and repository.

`cvs update [file ...]` Actualize the WA. Differences (if any) are `merged` into WA. Use this to incorporate changes from the repository.

Advanced commands

`cvs add [file/dir ...]` Adds files or directories into repository.

`cvs remove [file/dir ...]` Removes files or directories from repository.

`cvs rtag <tag> <module>` Adds a symbolic tag to the files of the module.

This list covers only basic functionalities, complete description can be found on the man page.