



# *Configuration Management with CVS*

Java-Praktikum SS 2002

Karsten Stahl    Martin Steffen

# Configuration management

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

# CM (cont'd)

## 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

## 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: of status (releases, baselines).
- Verification: guarantees completeness and consistency
- Construction: of the product from its constituents
- process management: support the software **life cycle**.
- teamwork: several teams/developers of one product

# CM informally

programming-in-the-many:  $\Rightarrow$  potential for confusion. CM is meant to decrease the confusion. CM must

- identify,
- organize, and
- control

changes 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: parallel development/test and parallel (sic!) changes of the same parts of the software
- Makefiles: construction and dependency checks

# Tasks of a CM system

Typical CM activities from the viewpoint of a developer (= user)

- check out current product parts
- build 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?

- parallel development
- remote development
- a modular extension of rcs
- free, robust, widely-used, and stable.



# CVS conflict resolution policy

- Developer works on **copy** from repository
- no “locking” <sup>a</sup>
- ⇒ parallel development
- ⇒ conflict: differing modifications of the same source.
- CVS keeps track of the dangers, **warns** the user
  - harmless merges: user is briefly **informed**
  - real conflict: user is **forced** to take decision

---

<sup>a</sup>in principle

# Basic CVS commands

(cf. also the [handout](#) and our web-page)

- general form: `cv`s command [options]  
[files..]
- `cv`s checkout <module>: gives the latest sources of <module> into the **current** directory; creates a **work-area**(WA).
- `cv`s commit [file ...] store back, asks for **comment**; fails, if WA is out of date.
- `cv`s diff [file ...]: look at changes
- `cv`s update [file ...]: bring WA into sync; potential differences are merged in

## (More) advanced commands

- `cv`s `add` [`file/dir...`]: add files or directories into repos
- `cv`s `remove` [`file/dir...`] removes files or directories from repository. <sup>a</sup>
- `cv`s `tag` <tag> <module>: adds symbolic tag
- much more can be done (**administrative** commands)
- CVS is **powerful**, use it **with care!**

---

<sup>a</sup>They can be recovered!

## *In our project: Rules of the game*

- check-in **compilable versions only!**
- don't interfere destructively with others
  - changes in other teams' packages: think twice, communicate
  - no un-announced **change of directory structure** (own subdirectories are ok)
  - hands-off the **administrative** cvs files
  - no **undo** of other people's changes without communication
  - no "*watches*" (except perhaps on own code)
- use `Makefiles` and `Readmes`
- use *Javadoc* + the cvs-logging mechanism

# In our project: Technical issues

- Repository:

```
swprakt@<machine>/home/swprakt/cvsroot
```

where <machine> is any of the pools computers, for instance `goofy.informatik.uni-kiel.de`

- root **module**: `Snot`
- necessary on client-side: `cvs` + `ssh` (secure shell)
- we need `ssh-keys` from everybody

For further info, see [handouts](#) and our web-page.

# References

- [CVS01] Concurrent versions systems: The open standard for version control. available at <http://www.cvshome.org/>, 2001.
- [Fog00] Karl Fogel. *Open Source Projekte mit CVS*. MITP-Verlag, 2000.