PVG (EDA260) - lecture 3: Konfigurationshantering

Lars Bendix

Department of Computer Science
Lund Institute of Technology
Sweden

What is SCM?

Software Configuration Management:
is the discipline of organising, controlling and managing the development and evolution of software systems. (IEEE, ISO,...)

The goal is to maximize productivity by minimizing mistakes. (Babich)

- Carlo’s lemon marmalade
- Citroën C3 fires

Building on sand?

CM is a CMM level 2 key process area

Req. | Design | Testing | Coding | QA

Software Configuration Management

SCM for XP development

Support for:
- handling source code
- collective ownership
- simple integration
- painless refactoring
- ease of testing
- effortless releasing
- handling document(ation)

Goals

- to be able to return to well-defined states
- to have an overview of the development history
- *to give a model for the system architecture*
- *to show what depends on what*
- *to ensure the consistent generation of a system*

- to save space
- to save time

An ounce of [history] is worth a pound of analysis.
Babich

How does a programmer spend his time?

- 50 % interacting with other team members
- 30 % working alone (pair-programming??)
- 20 % non-productive activities

Common heritage

- sharing things
- memory/history
- communication
- co-ordination

Problems of co-ordination

Shared data

Double maintenance

Simultaneous update

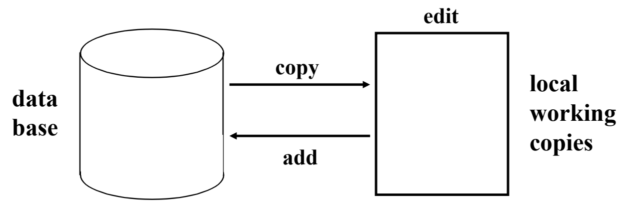
Co-ordination

Working in isolation:
- local dynamicity
- global stability
- problem:
  - multiple maintenance

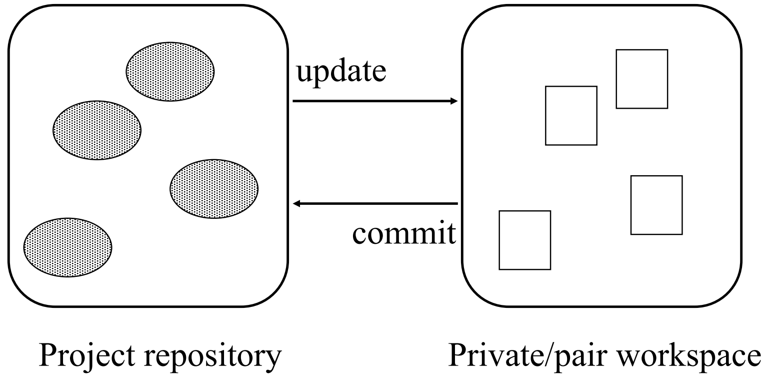
Working in group:
- global dynamicity
- problems:
  - shared data
  - simultaneous update

Immutability principle

Principle: components are **immutable**



Working



Copy/merge work model

Can we *lock* the things we want to work on? NO!

So we **copy** everything to our workspace...
...and everyone else copy to their workspaces...
⇒ double maintenance !!
o

Fortunately ”update” has a built-in **merge** facility:
- We first merge from the repository into the workspace
- Then we commit (copy/add) to the repository

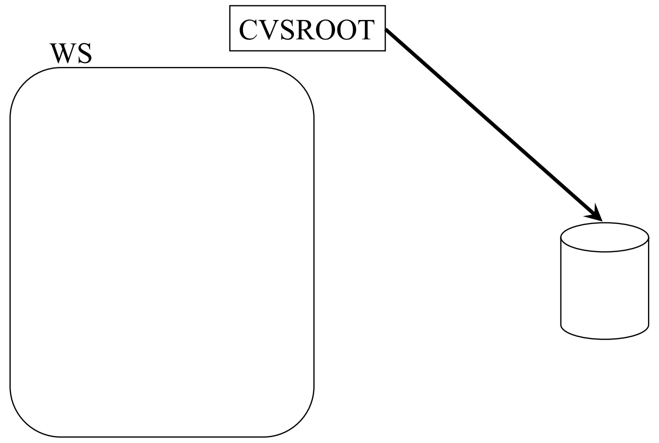
Quotes from XP'ers

- Overall CVS (and CM) was a HUGE help for the project.
- The version history was a real life saver.
- CVS made it possible for 12 people to work on the same code at the same time.
- CVS rules!
- It would have been impossible to merge different people's work without it.
- CVS sucks!
- Branching made releasing much easier.
- We tagged the releases – it served it's purpose.

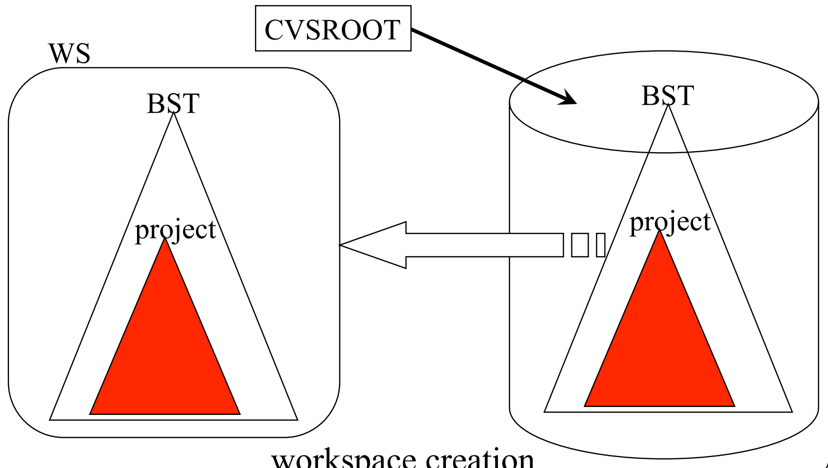
So how is CM used?

- update-commit
- merge – merge – merge
- no versioning, diff, tag, …
- change log only to identify people

mkdir WS; export CVSROOT=….

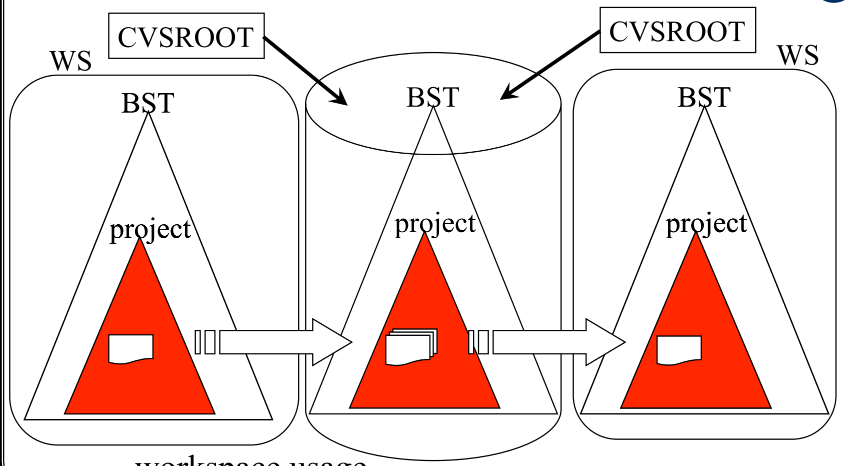


cvs checkout BST



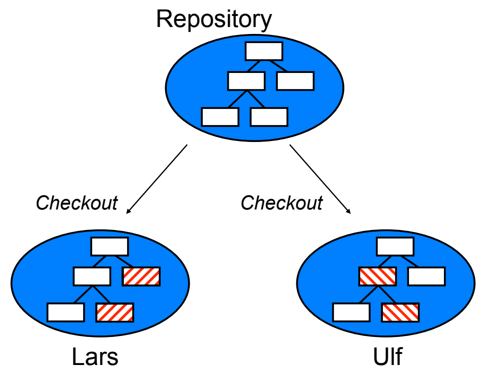
workspace creation

cvs status; cvs commit; cvs update;



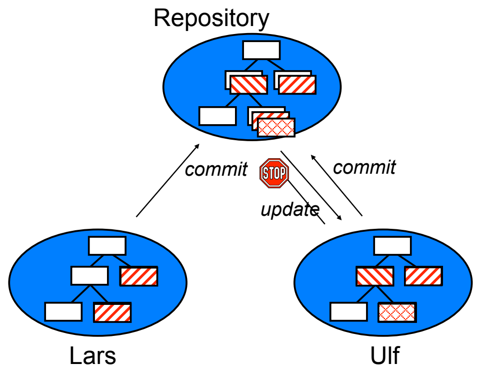
workspace usage

Long transactions I



workspace creation

Long transactions II



workspace usage (termination)

Extreme programming

SCM-related practices:
- collective ownership (developer)
- continuous integration (developer)
- refactoring (coding)
- small releases (business)
- planning game (business/developer)
- test-driven development (developer)

Collective code ownership

Goal: to spread the responsibility for the code to the team

How/why:
- from individual (pair) to team ownership
- reinforces code review (and readability)
- enables refactoring

Requires:
- team spirit
- frequent integration

SCM dangers:
- huge merge conflicts

Integrate continually I

Goal: to reduce the impact of adding new features

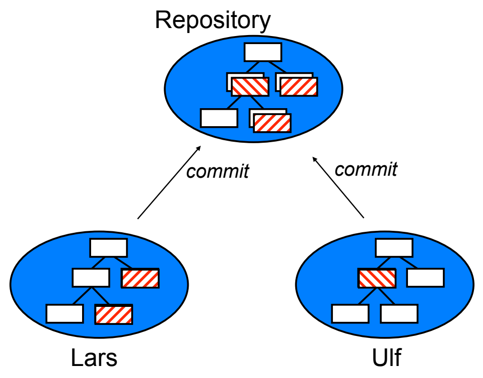
How/why:
- ”download” & ”upload” integration
- run tests; update (merge); re-run tests; commit
- ***all*** components must be in repository
- integration machine/responsibility/how often?
- keeps everyone in synchronisation
- keeps the project releasable all the time

Integrate continually II

Requires:
- collective source code repository
- short tasks

SCM dangers:
- huge merge conflicts
- false positives

Unfortunately :-(



NO ***strict*** long transactions - so...

Refactor mercilessly

Goal: to find the code's optimal design

How:
- before & after a task, think about refactoring
- changes the structure, but *not* the behaviour
- break out code; remove duplications; …

Requires:
- collective code ownership
- coding standards

SCM dangers:
- big-bang refactorings

Release regularly

Goal: to return the customer's investment often

Why/when/how:
- two-way feedback
- at the end of each iteration (daily?)
- clean machine principle
- automating and optimising the release

Requires:
- continuous integration

SCM dangers:
- a happy customer ;-)
- a broken release :-(

Play the Planning Game

Goal: to schedule the most important work

Why/how:
- to maximize the value of features produced
- divides planning responsibilities (what/how)
- developers estimate user stories
- developers split stories up into tasks

Requires:
- active customer
- mutual respect

SCM dangers:
- sloppy estimates and work break-down

XP process

1. Always start with all of the "released" code.
2. Write tests that correspond to your tasks.
3. Run all unit tests.
4. Fix any unit tests that are broken.
5. When all unit tests run, your local changes become release candidates.
6. Release candidate changes are integrated with the currently released code.
7. If the released code was modified, compare the differences and integrate them with your changes.
8. Rerun tests, fix, rerun tests, fix, rerun ….
9. When the unit tests run, release all of your code, making a new official version.

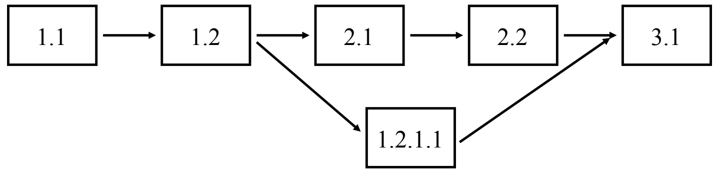
Diffing and merging

Visualisation of differences:

- diff

Merging of branches:

- merge
- always control manually that things went well



Log book

Gives the history for a component:

- who
- what
- when
- why

However, forget about the knowledge - as long as you can find the right person!

Code management tools

1. Identify local changes.
2. Differentiate between local changes and released code.
3. Identify who released a change and when they released it.
4. Merge changes and released code.
5. Revert to previously released code.

Troubleshooting

Slow merges

- “release” frequently
- don’t worry - be happy

Lost changes

- incorrectly merging
- intentional reversion
- wont stay lost for long

http://cs.lth.se/EDAN10/