# Introduction to Software Configuration Management Part I: The Team

Lars Bendix
bendix@sneSCM.org

*Scandinavian Network of Excellence in Software Configuration Management (sneSCM.org)*

*Department of Computer Science Lund University Sweden*

https://cs.lth.se/~bendix/Teaching/1-ECTS-SCM/



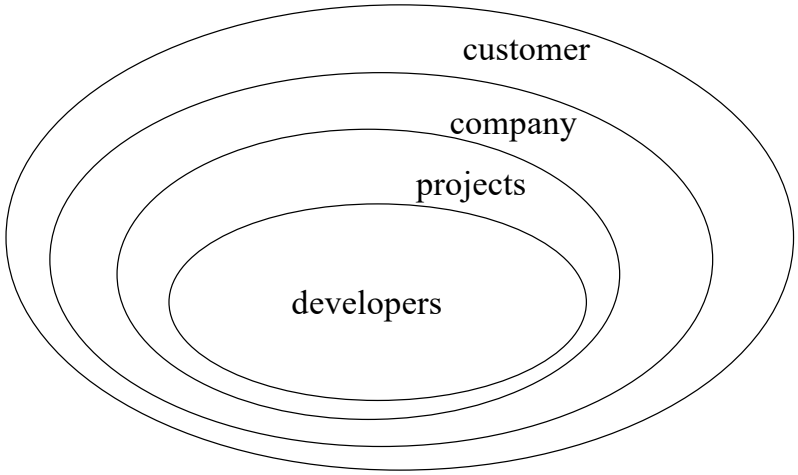
## Learning goals

After this first part the student will:

- understand coordination problems
- understand some important versioning concepts
- understand basic co-ordination strategies
- know some basic CM concepts
- be able to use a version control tool



## History of SCM






## 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 [programmer] productivity by minimizing [co-ordination] mistakes. (Babich)

# Problems

Identification:

You should be able to identify the single components and configurations.

- This worked yesterday, what has happened?
- Do we have the latest version?
- I have already fixed this problem. Why is it still there?

Change tracking:

Helps in tracking which changes have been made to which modules and by whom, when and why.

- Has this problem been fixed?
- Who is responsible for this change?
- This change looks obvious - has it been tried before?

# Problems

Software production:

Construction of a program involves pre-processing, compilation, linking, etc.

- I just corrected this, has something not been compiled?
- How was this binary produced?
- Did we do all the necessary steps in the right order?

Concurrent updating:

The system should offer possibilities for concurrent changes to components.

- Why did my changes disappear?
- How do I get these changes into my version?
- Are our changes in conflict with each other?

# Excuses for not using CM?

- CM only applies to source code
- CM is not appropriate during development because we use rapid prototyping
- It's not that big a project
- You can't stop people from making a quick patch
- We lower our cost by using only minimum-wage persons on our CM staff because CM does not require much skill ;-)

# How does a programmer spend his time?

- 50% interacting with other team members
- 30% working alone
- 20% non-productive activities

# (Software) development

- re-use things
- sharing things
- memory/history

- collaboration
- co-ordination
- communication

# SCM Hall of Fame

**Wayne Babich**, 1986:
*Software Configuration Management – Coordination for Team Productivity*

Team co-ordination problems:

- shared data
- double maintenance
- simultaneous update

"An ounce of [SBoM] is worth a pound of analysis"

# Wayne Babich I

Sometimes it is embarrassing to be a computer programmer. What other profession has such a remarkable rate of schedule and cost overrun and outright failure? [...]

Our failures are not of the individual contributors; most of us design, code and debug adequately or even well. Rather, the failure is one of coordination. Somehow we lack the ability to take 20 or 30 good programmers and meld them into a consistently productive team.

*Wayne A. Babich, 1986*

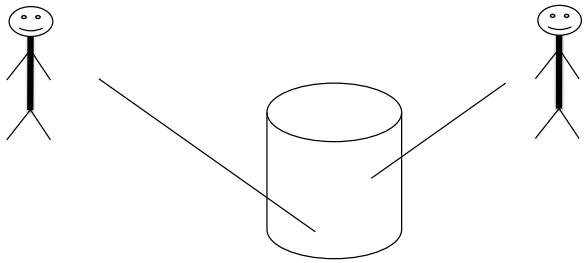
# Wayne Babich II

The term configuration management derives from the hard engineering disciplines [...], which use change control techniques to manage blueprints and other design documents. The term software configuration management has traditionally been applied to the process of describing and tracking releases of software as the product leaves the development group for the outside world.

I use the term in a more expansive sense. I include not just the formal release of software to the customer, but the day-to-day and minute-by-minute evolution of the software inside the development team. Controlled evolution means that you not only understand *what* you have *when* you are **delivering it**, but you also understand *what* you have *while* you are **developing it**.

*Wayne A. Babich, 1986*

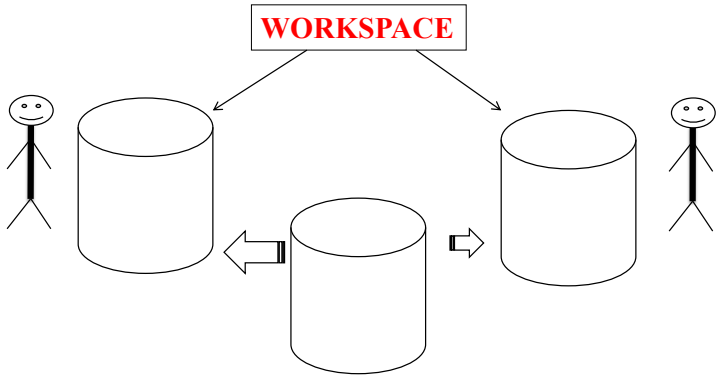
# Shared data



File server



# Solution 1



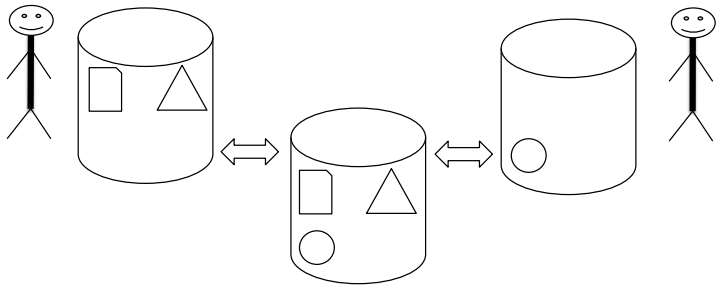

File server



# Double maintenance

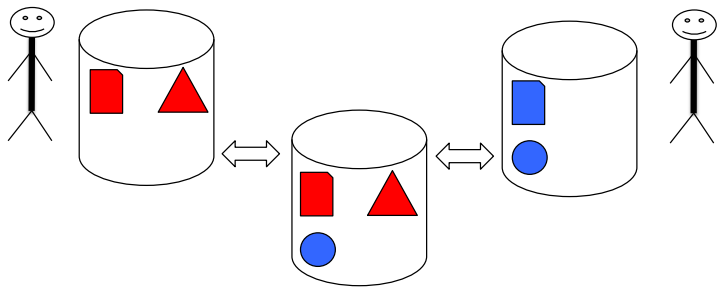
# Solution 2

"copy-write"



REPOSITORY

# Simultaneous update



REPOSITORY



# Co-ordination

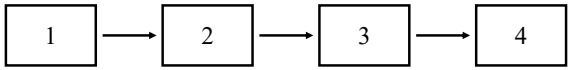
Working in isolation:
- local dynamicity
- global stability
- problems:
  - double maintenance

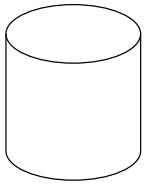
Working in group:
- global dynamicity
- problems:
  - shared data
  - simultaneous update



# Solution 3

"copy-add"

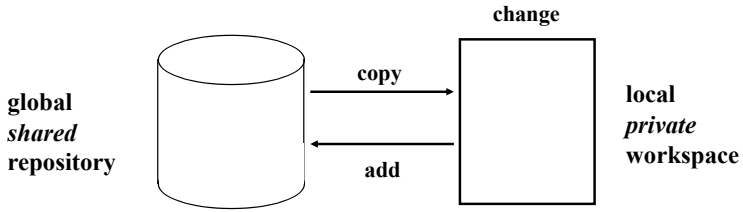


VERSIONING



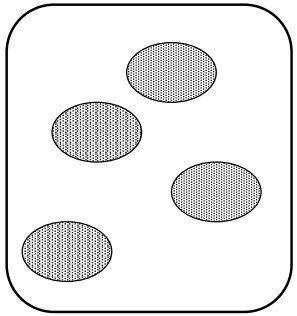
# Model



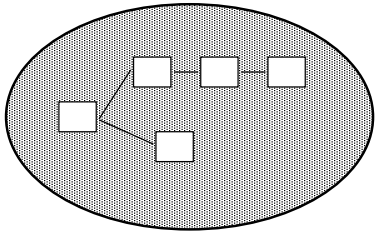

Principle: components are **immutable**

Babich:
- shared data
- double maintenance
- simultaneous update

## Concepts



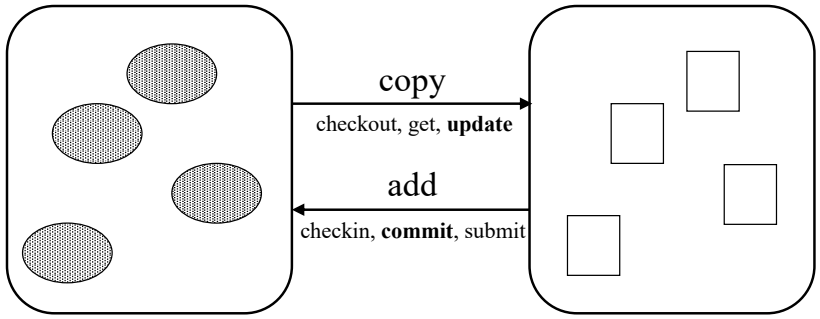
Repository



Version group



## Workspace/repository



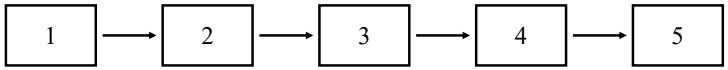

Project repository

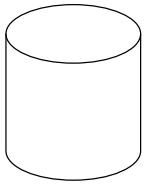
Private workspace



## Solution 3

"copy-add"



**Concurrency check**

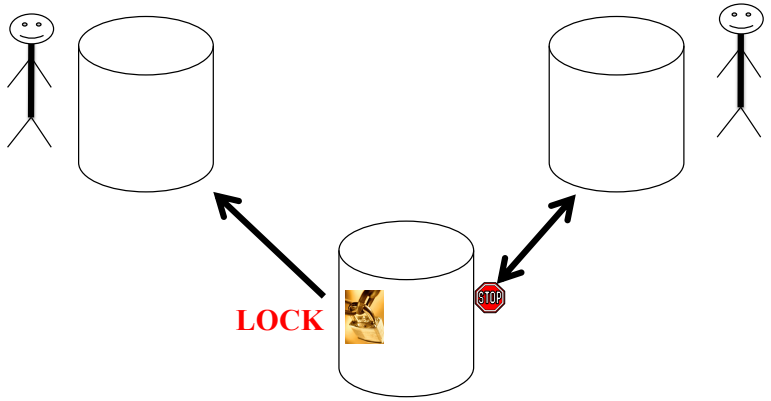


## Parallel work

Concurrency strategies:

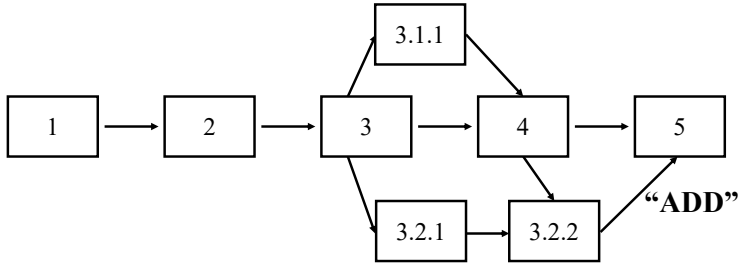
- pessimistic (locking)
- optimistic (copy-merge)

# Locking






# Branching






# Copy/merge work model

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

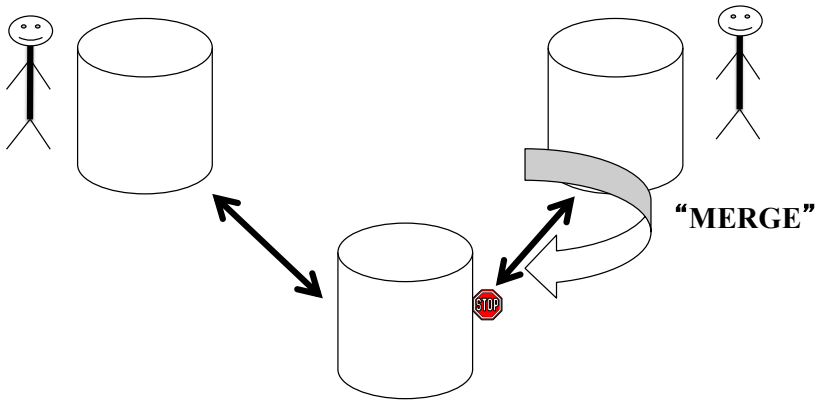
So we **copy** everything to our workspace ...
... and everyone else copy to their workspaces ...
⇒ double maintenance !!

Fortunately ”update” has a built-in **merge** facility:

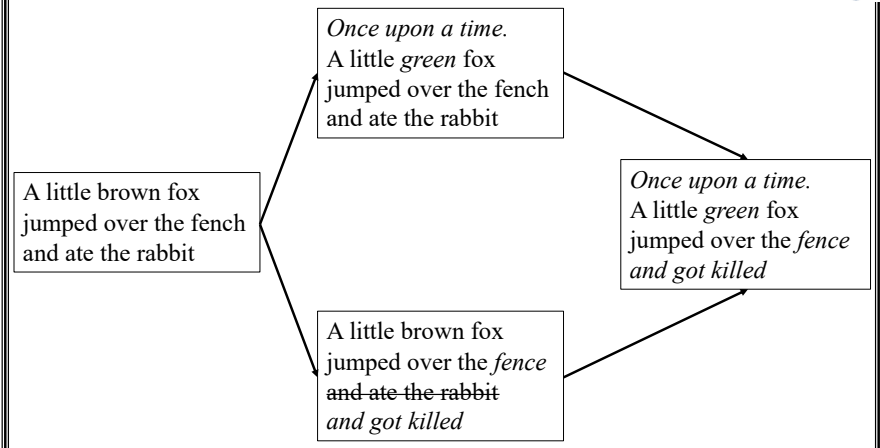
- We first merge *from* the repository *into* the workspace
- Then we check and fix problems
- Finally we commit (**add safely**) to the repository



# Copy-merge

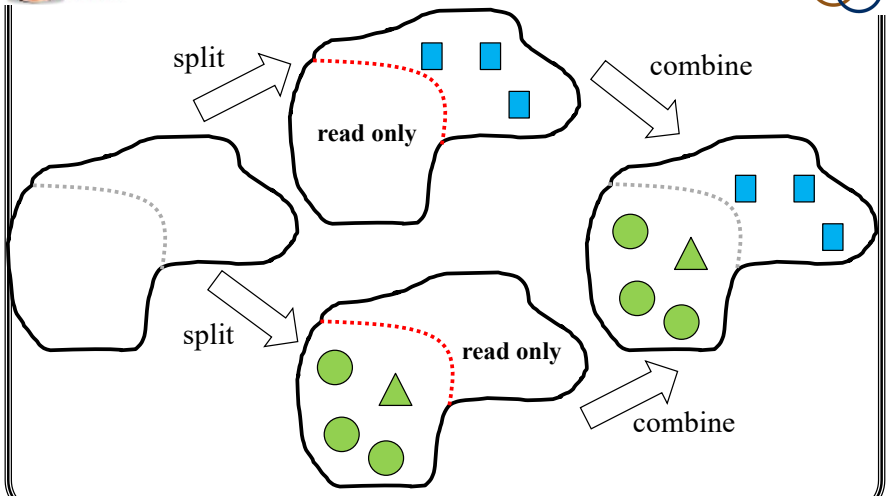


## 3-way merging






## Split-combine






## Long transactions

We do our change as a **logical unit**, we want this logical unit to be kept/recorded:

⇒ if there is anything that ”conflicts”, we must abort the **whole** commit (long transaction/atomic commit)

So far we have looked only at one type of ”conflicts”:

- people changing the same file in parallel



## Strict long transactions

But we are interested in whole **systems/configurations**:

- should we merge systems/configurations?
- yes, but **never** into the repository!

From physical conflicts to logical conflicts:

No new configurations should be created in the repo
New configurations should be created in the workspace
New configurations should be tested in the workspace
Then it should be “copied/added” to the repository

What can we do if the tool does not have SLT?

# Working together

I might still not want things to change:

- baseline (freezing a configuration)

I might want to know who is affected by my changes:

- traceability

I might want to know exactly how a system was built:

- software bill of material (SBoM)