



T-106.3101

Team work & Project Management, Version control

Jari Vanhanen/SoberIT

28.2.2008



Motivation - Software Development Scenario 1

- Small software
- Developed alone
- As a passionate hobby
- For the own needs of the developer
- No major consequences of bugs
- No schedule pressure
- No limitations on effort usage
- Software will be maintained by nobody or the developer himself





Motivation - Software Development Scenario

2

- Large software
- Developed by a team
- Used by many different users
- Software is done for a paying customer
- Every work hour costs money
- Management wants to follow the project
- Strict schedule and budget
- Bugs may cause serious consequences
- Maintained by others

What needs attention in this scenario?





Motivation - Software Development Scenario 2

- Large software (complexity, architectural design)
- Developed by a team (communication, coordination, team spirit)
- Used by many different users (understanding real needs)
- Software is done for a paying customer (accountability)
- Every work hour costs money (efficiency, prioritization)
- Management wants to follow the project (visibility, risks)
- Strict schedule and budget (predictability)
- Bugs may cause serious consequences (quality, proof of quality)
- Maintained by others (maintainability, documentation, training)





Team Work & Project Management



A story of four people...

This is a story about four people named Everybody, Somebody, Anybody and Nobody.

There was an important job to be done and Everybody was sure that Somebody would do it. Anybody could have done it, but Nobody did it.

Somebody got angry about that, because it was Everybody's job. Everybody thought Anybody would do it, and Nobody realized that Everybody wouldn't do it.

It ended up so that Everybody blamed Somebody when Nobody did what Anybody could have done.



Special Challenges in Student Projects

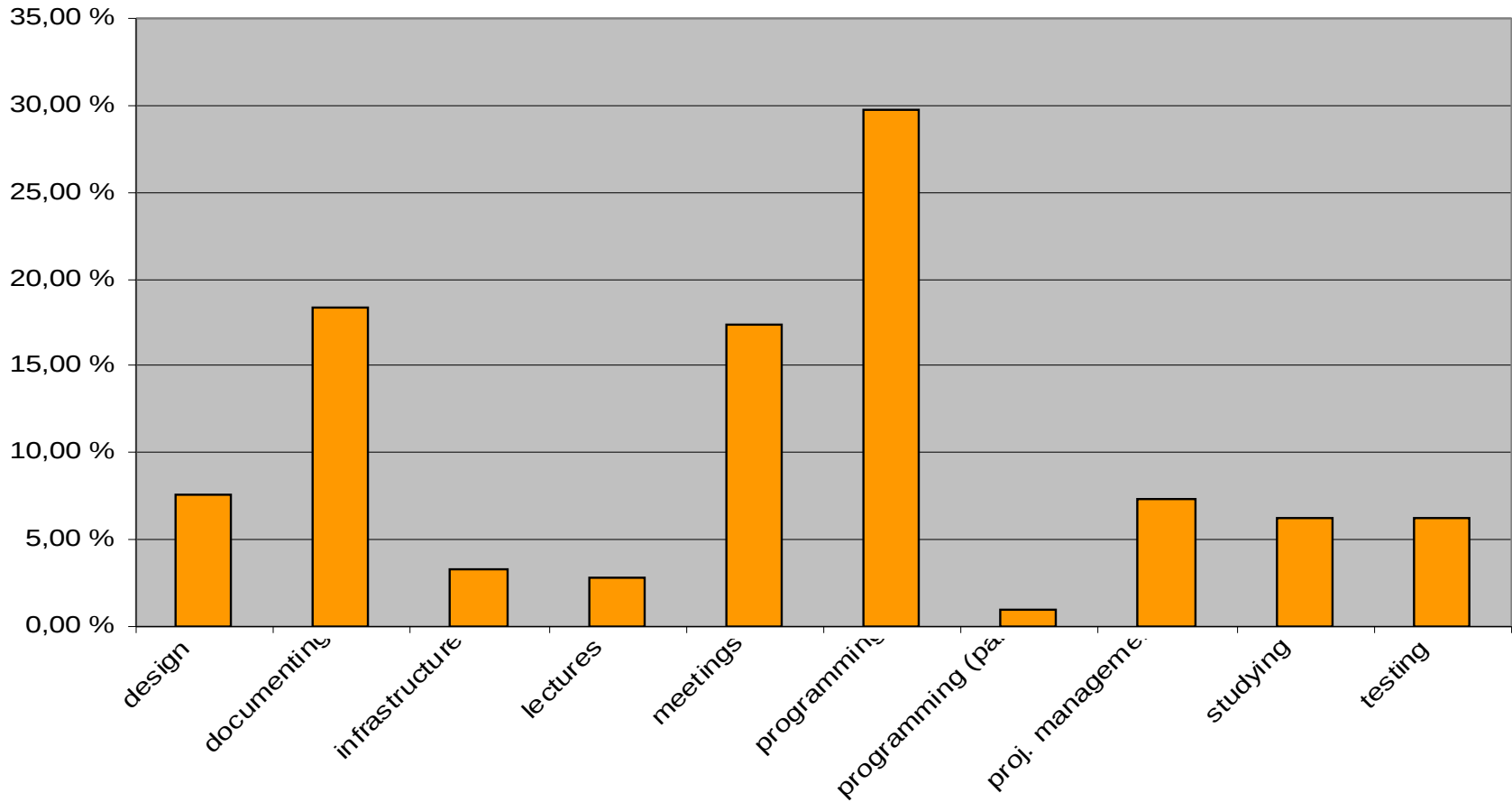
- Physical distribution
 - the group does not do all work in the same place
 - **special care for communication and coordination**

- Temporal distribution
 - only one of several on-going "projects" for all participants
 - long duration, part time work
 - **writing things down in order to recall them**

- Team has no existing development culture (process)
 - and people are not necessarily even familiar with each other
 - **work practices must be planned from scratch**
 - **team spirit and motivation**



Effort distribution in large student projects (~1200h)



source: T-76.4115 course projects 2004-05



Is project planning difficult?

- Probably, but it should provide answers to fairly simple questions:

- Why?	—————	Goals
- What?	—————	Work Products and Milestones
- When?	—————	
- Who?	—————	Responsibilities, Coordination
- Where?	—————	
- How?	—————	Approach, Practices, Tools
- How much?	—————	Resources
- Assuming?	—————	Risks, Beliefs, Priorities



Plan Roles

- Is someone the project manager?

- Principles for dividing the work
 - sub system based division
 - activity based division



Plan Phases

- Split the project into phases
 - rough goals
 - start and end dates

- Plan each phase in detail when it starts
 - goals and deliverables
 - tasks
 - schedule

- Phases provide control points for analyzing the progress



Plan Phases – Goals and Deliverables

- Define the general goals for this phase
 - “plan the project, understand the main requirements of the system, and set up the project infrastructure”
 - “implement the GUI and the highest priority features for a demo at COMDEX expo on 20.3.2007”
 - ...

- List what needs to be delivered
 - sw features
 - project and product documentation



Plan Phases – Tasks and Resources

- Derive the needed tasks for producing the deliverables
 - design, coding, testing, documenting
- Non-development tasks
 - project management, status meetings, studying etc.
- Assign the tasks
 - each task should have someone responsible for it
 - roles?
- Estimate task efforts
 - it is easier to estimate smaller tasks
 - base the estimate on previous experiences
 - collect more experience by personal estimation and tracking
- Reality check between the required and available effort
 - adjust scope or effort if necessary



Plan Phases - Schedule

- Identify possible dependencies between tasks
- Schedule tasks
 - leave buffer
- Set internal milestones
 - controlling and ensuring progress
 - work often gets done just before the deadlines



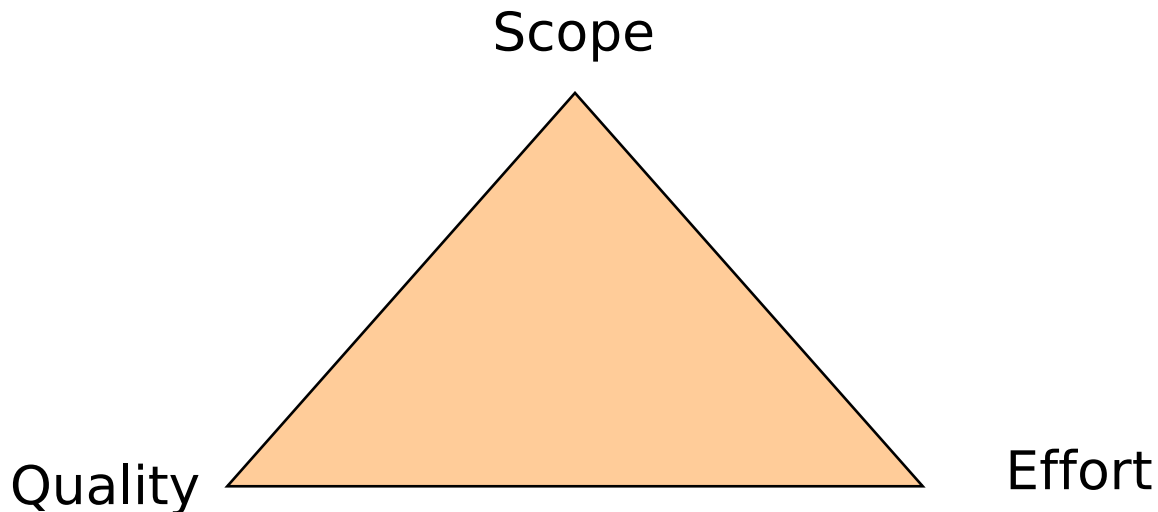
Plan Practices and Tools

- Project management
 - planning
 - progress tracking
 - risk management
- Design
 - understanding the existing system
 - strategy for implementing the new features
 - certainly an iterative activity
 - how it is done (together/by a lead architect?)
 - some design decisions must be know by the whole team
- Coding
 - development environment
 - managing simultaneous work
 - version control
 - minimizing dependencies between coders
- Testing
 - planning, doing, reporting (how, what, when, who,...)
 - defect management



Project Tracking and Control

- Project plan is a tool for tracking the project
- Plan never realizes perfectly
 - one must handle differences to the plan
- Project control variables





Communication

- Challenging for a distributed project team
- Information to share
 - project status
 - design decisions
 - problems
 - coding tips
 - ...
- Communication practices and tools
 - co-located team
 - face-to-face meetings
 - phone/Skype
 - irc
 - e-mail lists/discussion forums
 - Wiki
 - version control system
 - documents (product, project)
 - code comments

Having a common vocabulary for the project helps communication



Pair Programming (PP)

- Two developers, one task, one computer
- Driver and navigator roles
 - roles are switched
- Continuous collaboration
 - both must understand what is happening
- For all development activities
 - analysis, design, programming, testing
- Partners are changed periodically
 - e.g. after finishing a task



(Williams & Kessler 2002)



Proposed benefits and costs of using PP

BENEFITS

- Quality
 - less defects in code
 - better design solutions
 - comprehensible design and code
- Elapsed time
 - pair finishes tasks faster
 - shorter projects by adding people
 - more accurate effort estimates
- Human factors
 - satisfaction and confidence on work
 - trust and teamwork
 - more discipline in work
- Knowledge transfer
 - learning
 - fewer critical experts
 - teaching new employees quickly

COSTS

- Effort
 - learning time
 - learning PP
 - familiarizing with a new partner
 - increased effort on task level
 - ... but in the long term?
- Exhaustiveness
 - intensive way of working

In practice the effects seem to depend on the details of how PP is applied.



Project Retrospection

- Analyze the project in order to learn for the future
 - what worked well, what didn't work
 - why (consider the context of the project)
- Work practices and tools
- Estimated and realized effort



Hints

- Arrange a kick-off
 - get to know each other
 - agree on your team's goals
 - agree on practices and practicalities
- Start immediately
 - leaves more time to react to unexpected situations
- Try co-located team sessions
 - problems can be addressed immediately
 - peer pressure as booster for productivity
 - prepare well
 - hw & sw
 - pizza & coke

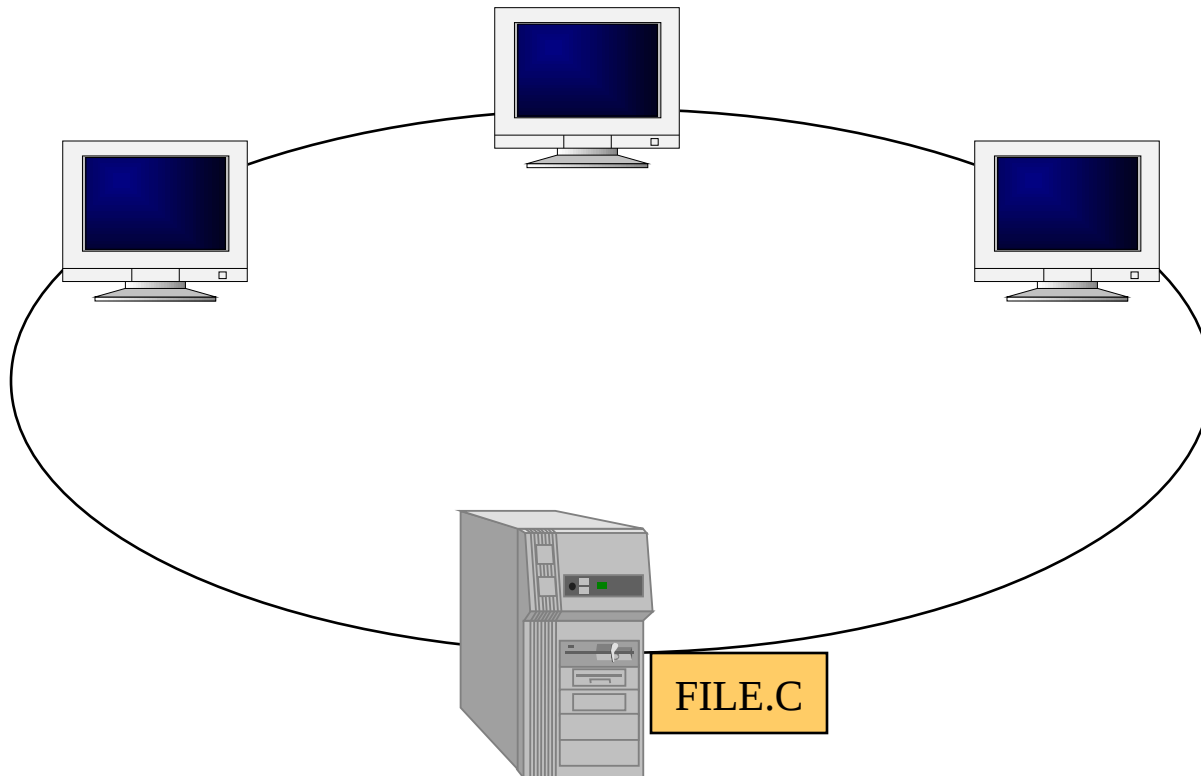


Version Management



Shared data problem

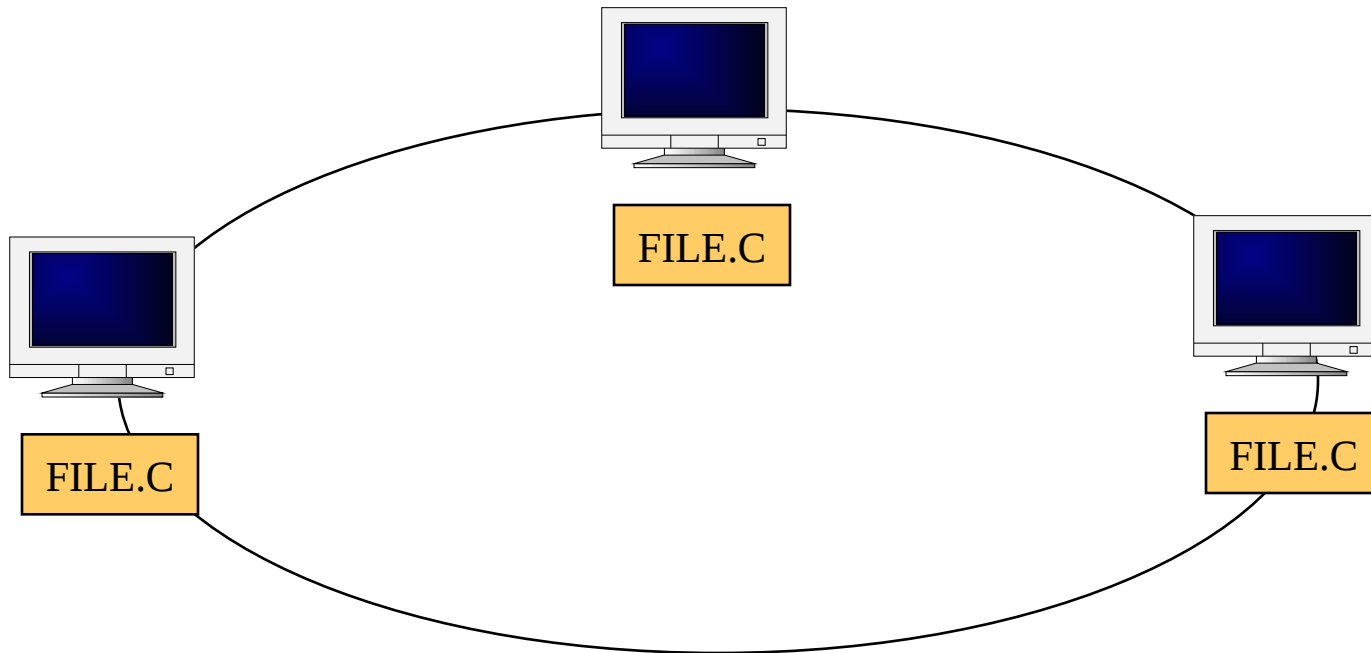
- File in one place
 - only one person is allowed to work with the file at a time





Double maintenance problem

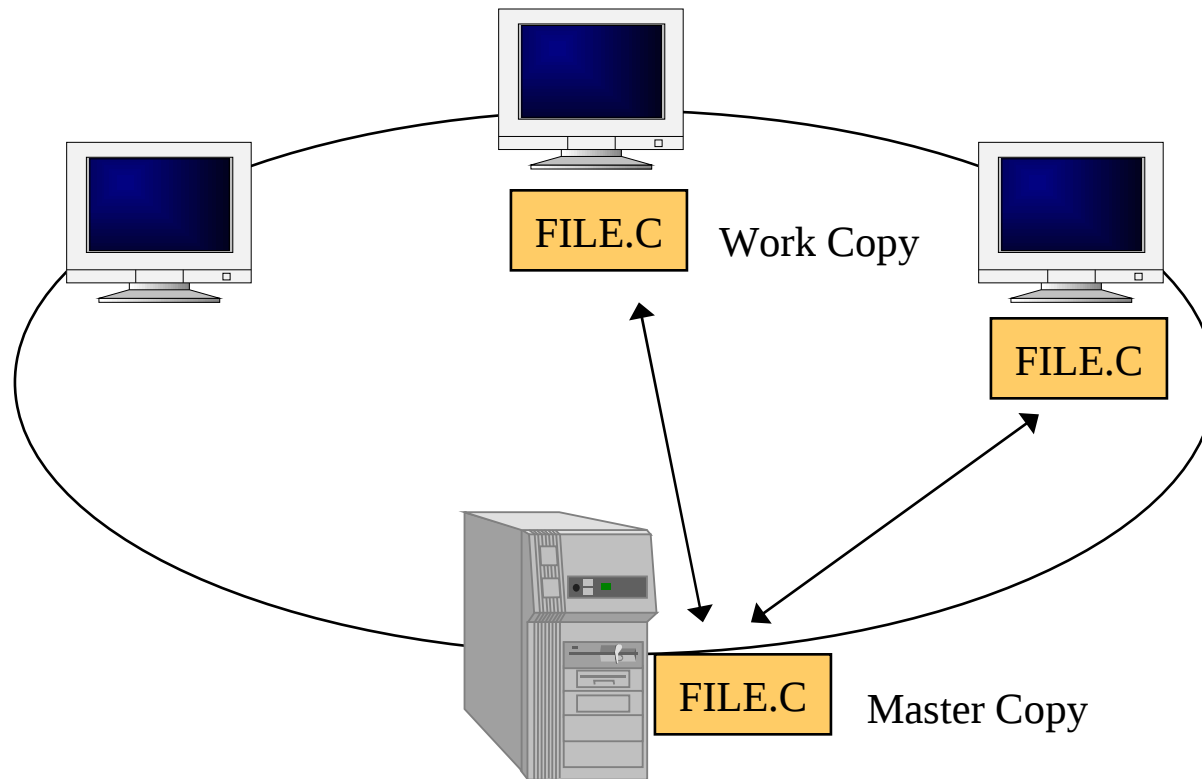
- Multiple copies of a file
 - when one is changed, all must be updated





Simultaneously update problem

- A master copy and personal work copies of a file
 - if two work copies are changed simultaneously, the latter changes overwrite the former changes





More version management challenges

- Old versions are needed sometimes
 - fixing previous releases
 - finding out what change broke the system
 - reverting to the latest working version



What needs to be under version control?

- Preferably everything
 - source code
 - requirements
 - design documents
 - scripts
 - test material
 - instructions
 - program builds (executables)
 - used external files (reused libraries)
 - used development tools (compiler)

“Configuration items”

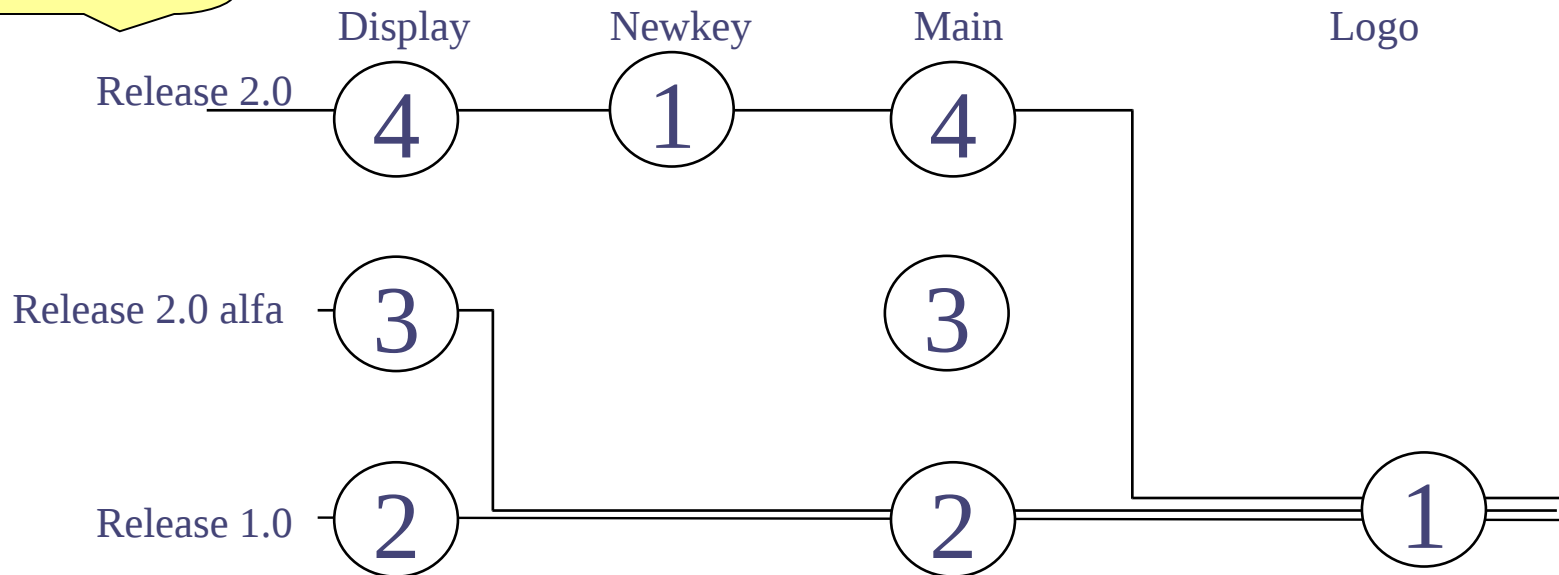
- typically files
- some tools support managing logical items such as requirements, classes, test cases



Configuration

- Certain versions of the *configuration items* having some logical purpose
 - e.g. product release 2.0

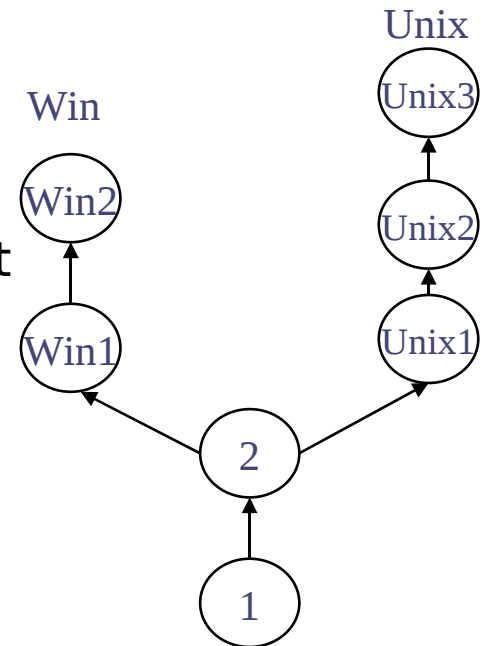
“tagging”





Two types of versions

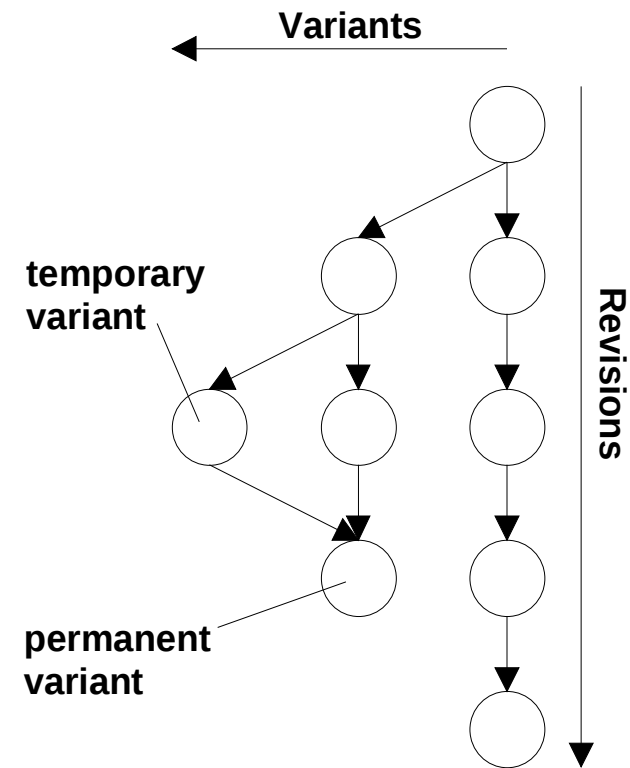
- Version
 - An instance of an item that in some way differs from other instances of the same item
- Variant
 - a version that is customized for a specific purpose
 - platform, customer
 - an alternative to another variant
 - “branch”
- Revision
 - a different version of one and the same variant
 - replaces the previous revision





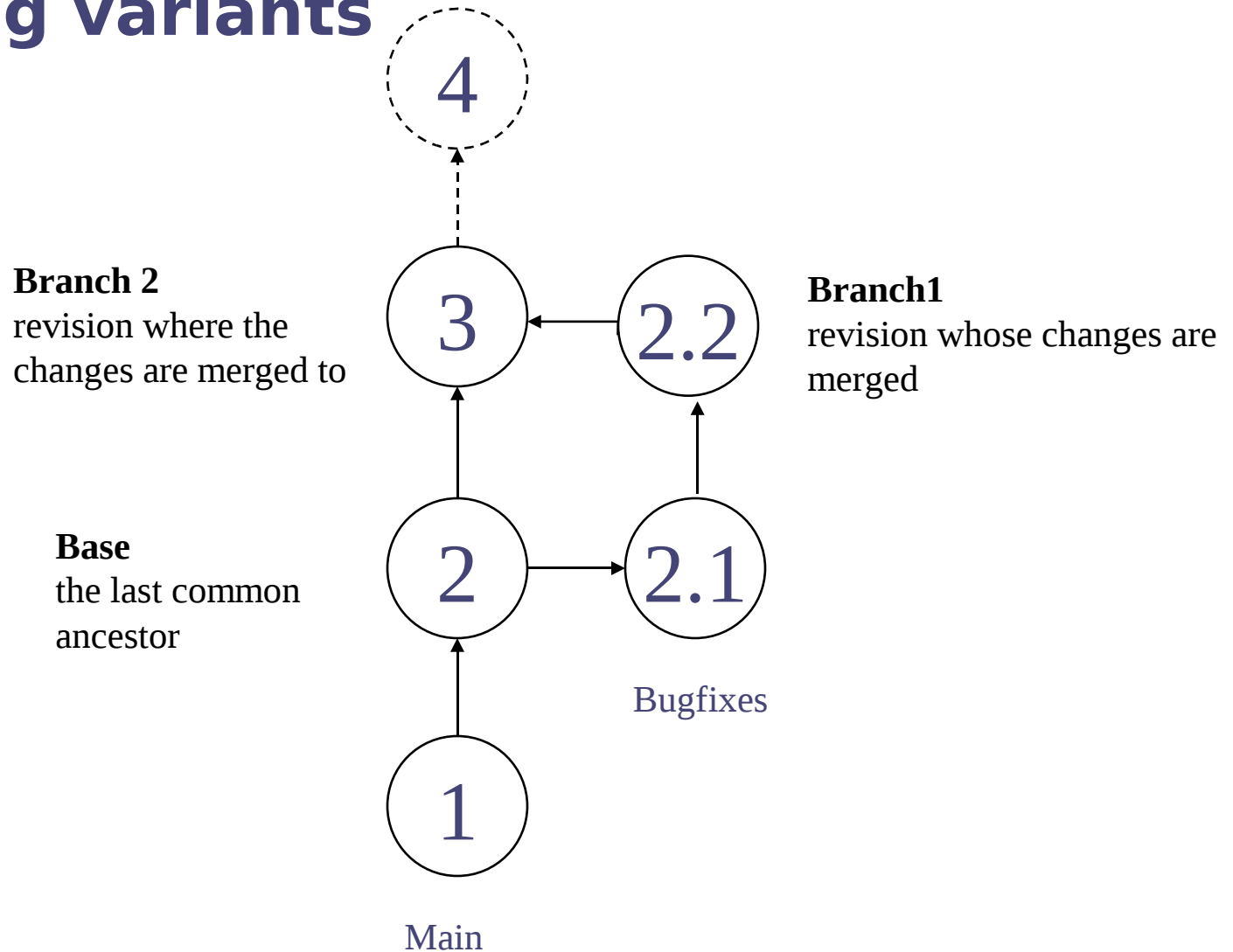
Two types of variants

- Permanent variants
 - changes to the variants will not be merged
 - e.g. a variant for each operating system
- Temporary variants
 - changes to the variants will be merged later
 - bug fixes
 - parallel development versions





Merging variants






Support for merging variants

- Version management tools can usually merge ASCII files
- Binary files typically require application specific tools



Textual merging

- Variants are compared line by line to a common ancestor
 - a change can add/delete/change a line
 - listing of diffs
- Non conflicting changes can be merged automatically
- Conflicting changes are marked
 - a tool **cannot** detect logical conflicts

		Branch 1	
		=Base	<>Base
Branch 2	=Base	=Base	=Branch 1
	<>Base	=Branch 2	 Conflict



Managing variance

- Localizing variance to as few items as possible
 - e.g. design a common interface for operating system specific calls
- Minimizing duplicated code between variants of a file
- Storing many variants within one source file
 - preprocessing #IF macros
 - variance is not localized
 - variants are not separated, independent development hard
 - files become unreadable



CVS

Concurrent Versions System

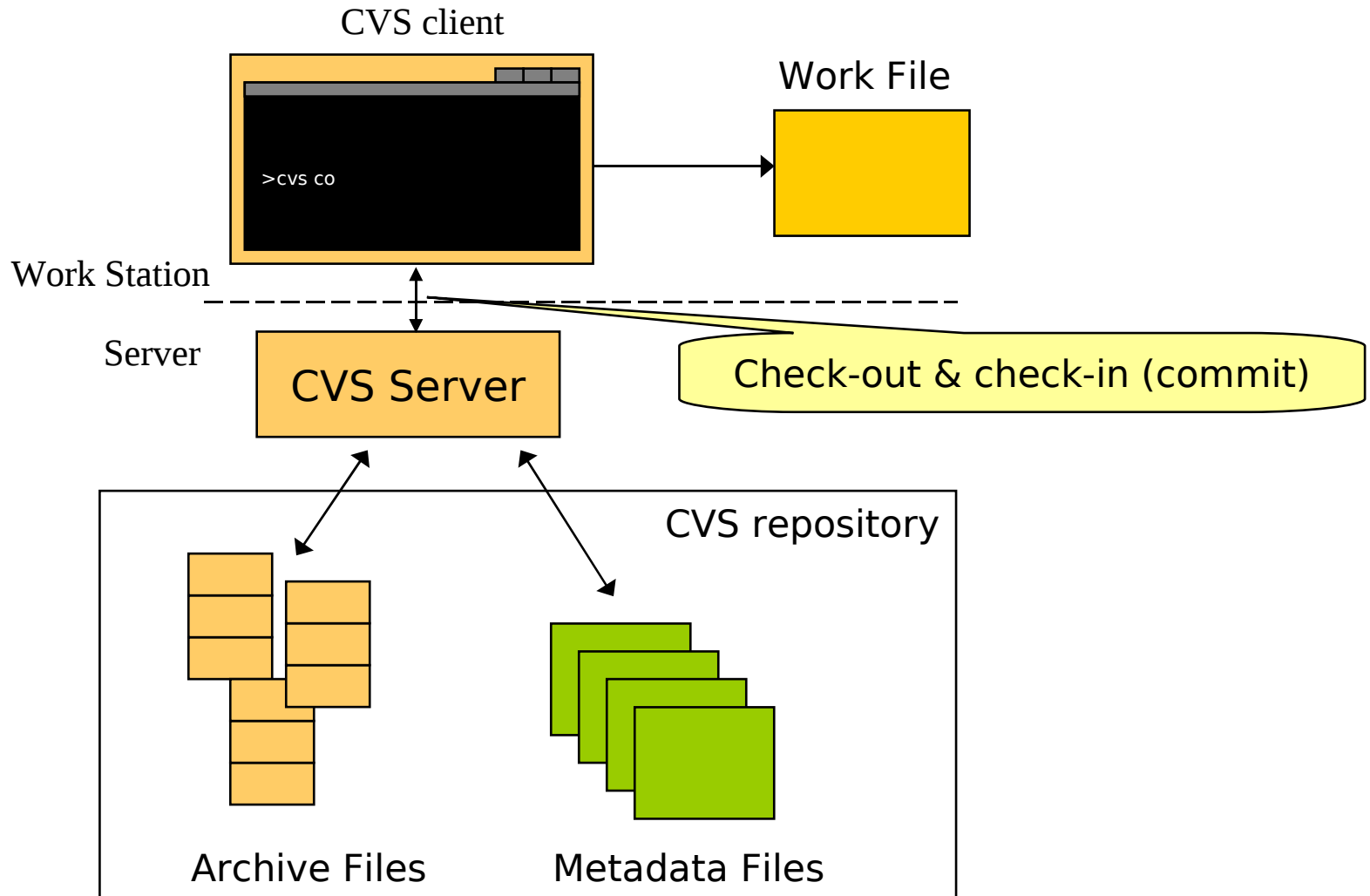


Introduction

- Open source
 - Available for all common operating systems
- Simple version control tool with some support for team work, product structure, and customization
- Client/server support
 - a single repository used by all clients
- Commands
 - `cvs [-cvs_opts] cmd [-cmd_opts] [cmd_args]`

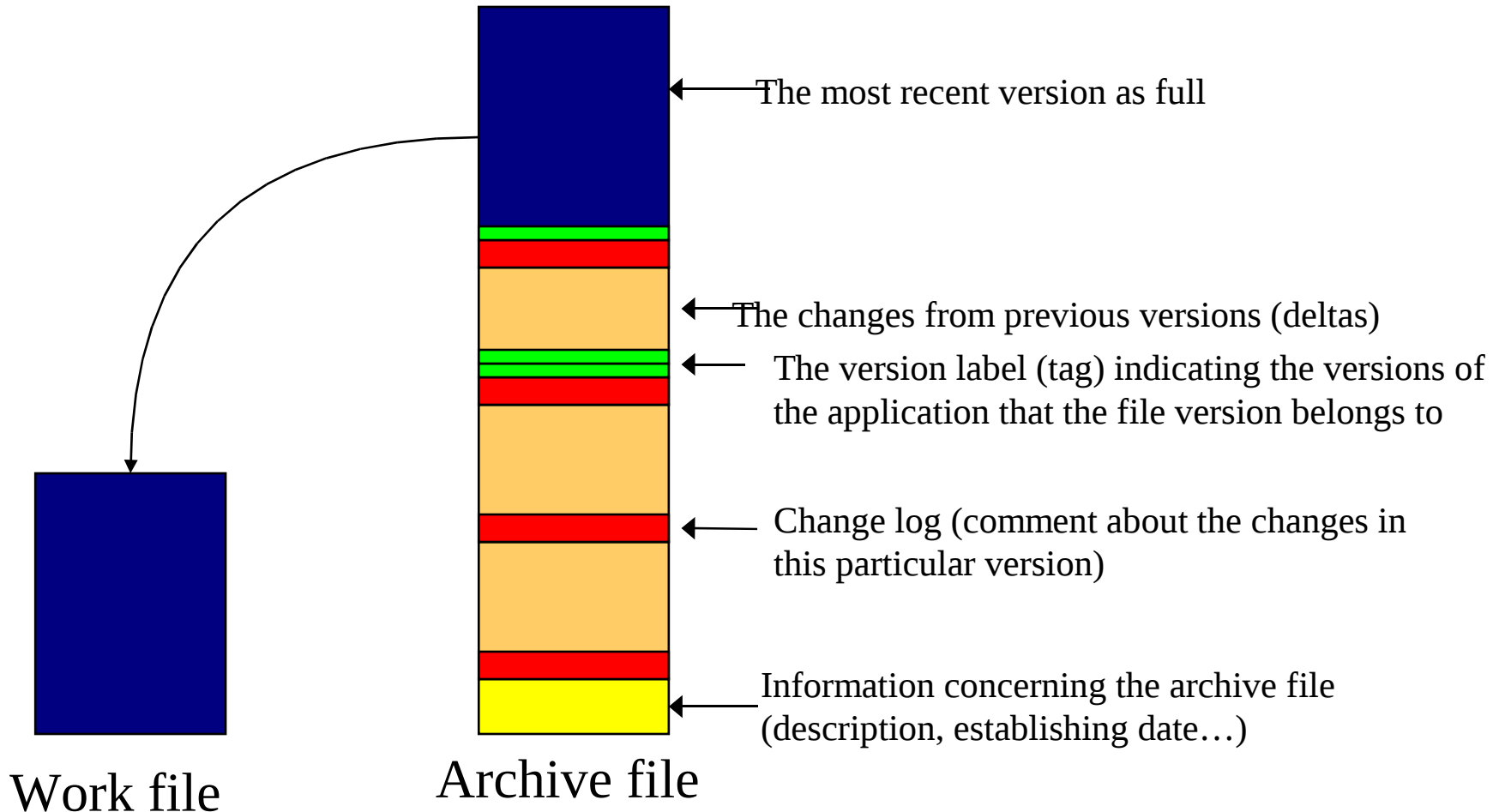


Architecture





Archive file





Repository

- CVSROOT directory contains administrative files
- Other directories contain history files (*,v)
 - a history (RCS) file contains all versions of a file and metadata
- Location specified by -d option or \$CVSROOT variable
 - `CVSROOT=:ext:user@vipunen.hut.fi:/.../mycvs/`
 - `CVS_RSH=ssh`
- Creating a repository
 - `(mkdir /.../mycvs)`
 - `cvs init`



Starting a project

- Alternatives for starting
 - import existing files
 - `cd wdir`
 - `cvs import myproj vendor start`
 - start from scratch
 - create an empty directory structure and import it to CVS
 - add files later using `cvs add` command



Basic usage

```
$ cvs checkout myproj  
$ cd myproj  
$ ls  
CVS Makefile backend.c driver.c frontend.c parser.c  
$ emacs backend.c driver.c  
$ cvs commit -m "Implemented feature xxx"
```

- A team should agree on check-in policies
 - commit after finishing a logical change
 - avoid committing broken code
 - disturbs the work of others
 - must compile, must pass unit tests, ...
 - write understandable change log comments
- Use "\$Log: \$" keyword to automatically show change logs in files



Multiple developers

- `cv`s `commit` fails if any of the committed files has changed in the repository
 - someone else has committed changes first
- `cv`s `update` merges recent revisions in the repository to those in the working directory
 - conflicts are marked and must be solved before a new commit
- `cv`s `diff` shows the differences between two versions



Tagging

- File versions don't match with product version numbers
 - different amount of versions of different files
- `cvstag rel-1-0 .`
 - assigns a tag to the current revision of all files in ./
- `cvsc checkout -r rel-1-0 myproj`
 - retrieves the rel-1-0 of module myproj
 - sets rel-1-0 to be the default branch in this work directory



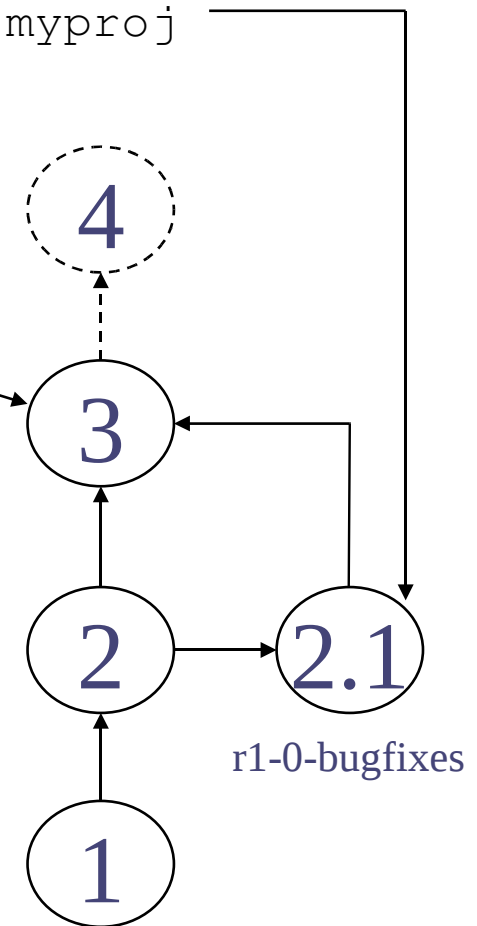
Branches and merging

■ Creating a branch

- `cv s rtag -b -r rel-1-0 rel-1-0-bugfixes myproj`

■ Merging

- `cv s checkout myproj`
- `cv s update -j r1-0-bugfixes files`
- `cv s commit -m "Included r1-0-bugfixes"`





Common problems

- Renaming/changing directories in the repository
- Binary files
 - remember to turn off keyword substitution



More information

- http://en.wikipedia.org/wiki/Concurrent_Versions_System
- See also Subversion
 - <http://subversion.tigris.org/>
 - “a compelling replacement for CVS”