

Project Development (SYP)

Configuration Management (CM)

Non-linear Process

Problem

Phenomena

Counter Measures

Project development
is a NON-linear
process

Wrong decisions

Bugs

Document
changes (when,
who, why)

Switch back to
old versions

Team Work

Problem

Phenomena

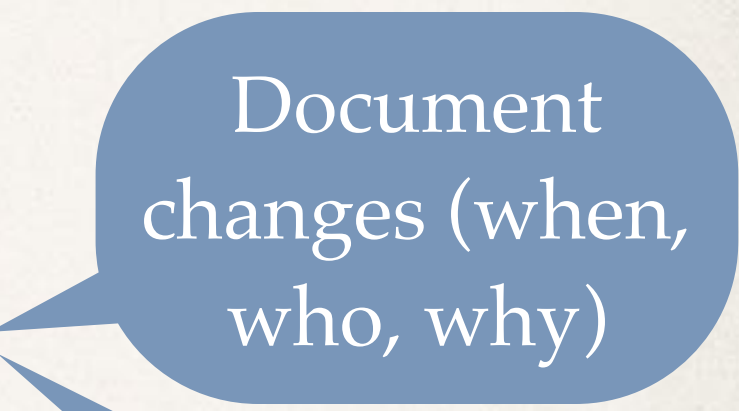
Counter Measures




Project development
is team work

Parallel work

... on different
files



Document
changes (when,
who, why)



Share changes
easily

Manage Different Versions

Problem

Proj. dev. is
management of
different versions

Phenomena

Stable releases

... while
development
continues

Variants needed

Counter Measures

Control
development
and integration

Control variants

Aim of Configuration Management

Analysis

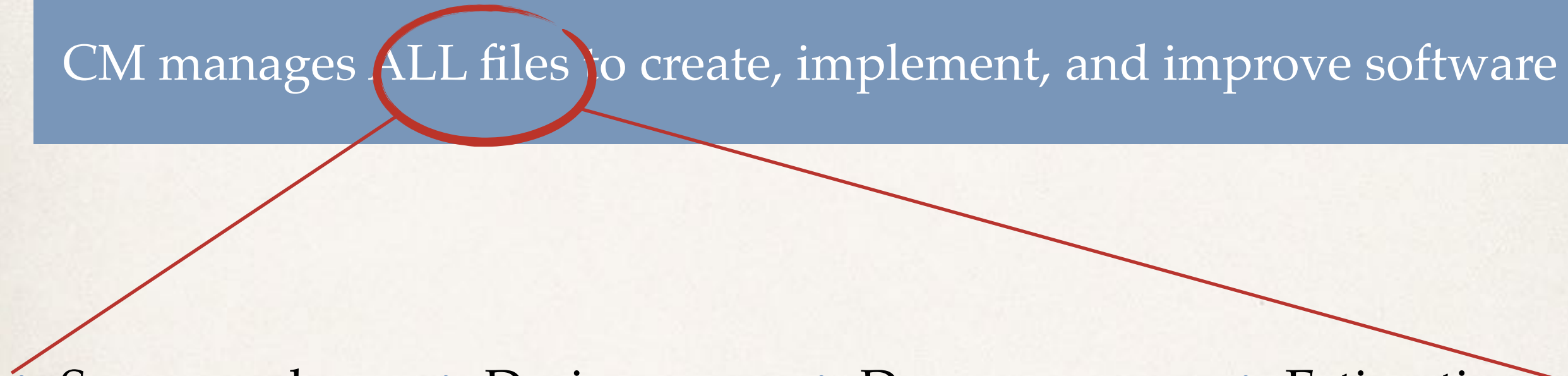
Design

Implementation

Test

Maintenance

CM manages ALL files to create, implement, and improve software

- 
- * Source code
 - * Designs
 - * Docu
 - * Estimations
 - * Build scripts
 - * Test scripts
 - * Specs
 - * Schedules
 - * Requirements
 - * Images
 - * Guidelines
 - * ...

CM Activities

- ❖ Identify artifacts to be controlled
 - ❖ Which files will be generated?
 - ❖ How will they be organized?
- ❖ Variant management
 - ❖ Which / How many versions do co-exist?
 - ❖ Which branches to maintain?
- ❖ Release Management
 - ❖ Baseline your development
 - ❖ Create versions for customer
- ❖ Change Management
 - ❖ Where to store change requests (CRs)?
 - ❖ Why and when to implement a CR?

Content of a CM Plan

- ❖ Versioning methods & tools
 - ❖ Which tools do you use? svn, git
 - ❖ Do you have all documents under version control?
 - ❖ If not, why not and how do you monitor changes?
- ❖ Organization of artifacts
- ❖ Variant management
- ❖ Release management
- ❖ Change management

Organization of Artifacts – Proposal

01_ProjectPlanning: Effort estimations, schedules

02_Requirements

03_Source Code

04_Testing

05_ReleaseDocs: Feature Lists, Change Logs, ...

06_Meeting Minutes

07_Templates

08_MiscDocs: Design Docs, Project Proposal, Customer Info, ...

Variant Management

- ❖ How to organize different versions and branches?
 - ❖ Where are different versions stored?
 - ❖ When to open a new branch?
- ❖ How to name new project versions?
 - ❖ Easy to remember: Yosemite, Nougat, etc.
 - ❖ Structured: v5.4.12
 - ❖ Mix of the above variants?

Working with Subversion

Possible Server

❖ subversionneu.htl-leonding.ac.at

Organization of Tags and Branches

- ❖ /trunk
- ❖ /tags
- ❖ /branches

Version Control with SVN – trunk, tags, and branches

- ❖ Three directories which are created by convention
- ❖ trunk
 - ❖ Means the main development line
- ❖ tags
 - ❖ Holds all frozen states of the project's artifacts
- ❖ branches
 - ❖ Holds development lines of project variants

Tag Names

- ✧ Tags are basically aligned with project milestones
- ✧ <PROJECT_ALIAS>_xx.yy.zz
 - ✧ Example: SIRIUS_01.00.00
- ✧ In svn this is a smart copy from /trunk into
 - ✧ /tags/<PROJECT_ALIAS>xx.yy.zz

Version Control with SVN – Terminology

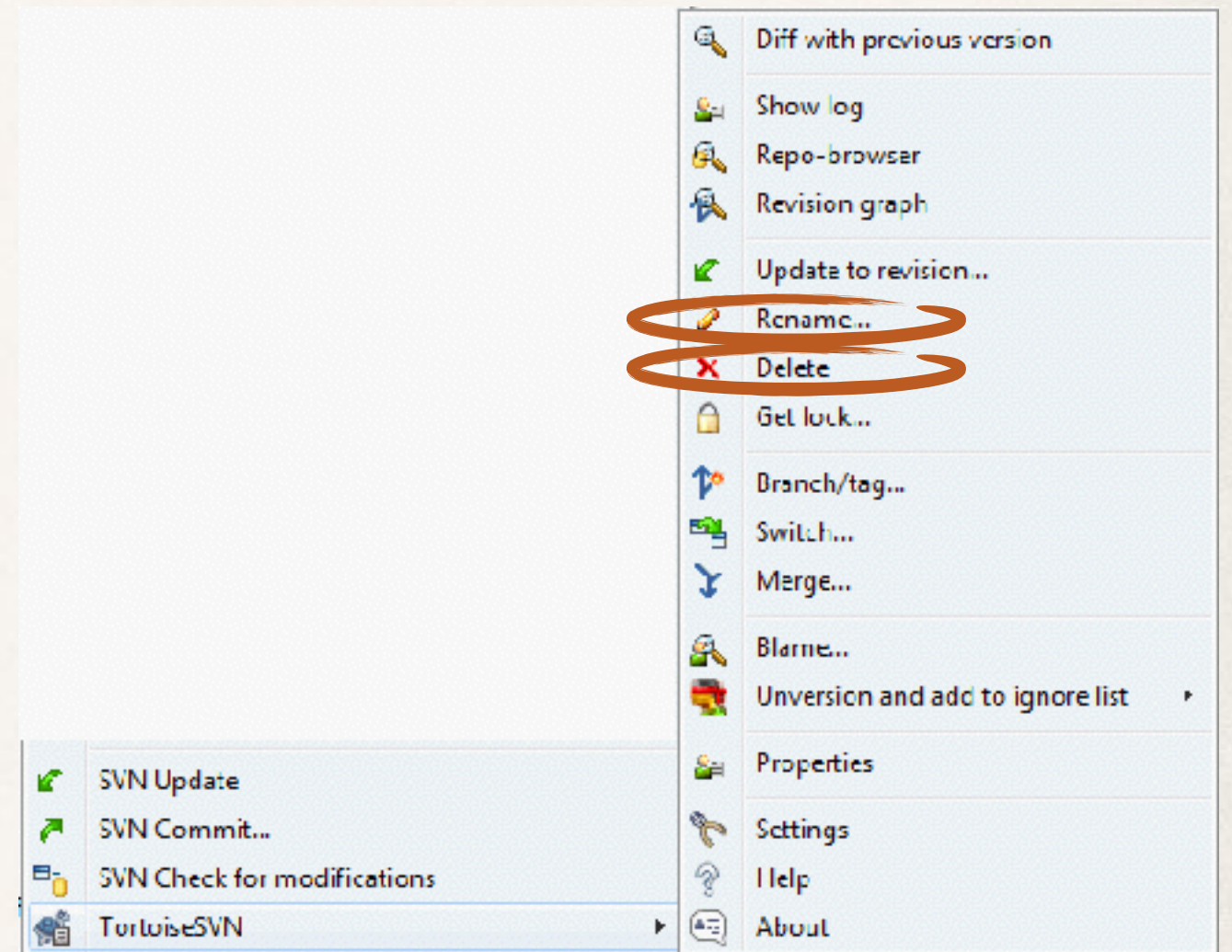
- ❖ Repository
 - ❖ Central store on server holding all artifacts of your project
- ❖ Working Copy
 - ❖ Local “sand-box” on individual computer
- ❖ Revision
 - ❖ State of a project’s artifacts in the repository

Version Control with SVN – Basic Operations

- ❖ Check-out
 - ❖ Initial transfer of project artifacts from repository to working copy
- ❖ Commit
 - ❖ Storing back changed artifacts from working copy to repository
- ❖ Update
 - ❖ Synchronize working copy with latest revision of repository

Version Control with SVN – Characteristics

- ❖ Never rename a file without svn in a working copy
 - ❖ `svn rename <file>`
- ❖ Never delete a file without svn in a working copy
 - ❖ `svn rm <file>`
- ❖ Never move a file without svn in a working copy
 - ❖ `svn mv <file>`



Right drag

Working with git

Possible Servers

- ❖ GitLab: <https://gitlab.htl-leonding.ac.at>
 - ❖ preferred – check out if working and sign-up if available
- ❖ GitHub: <https://github.com>
- ❖ BitBucket: <https://bitbucket.org/account/signin/>

Optional Exercise: Play Around With Git

Create a New Repository

- ❖ New repository, Name == 3BHIF_Test_Rep
 - ❖ Initialize with README.md
- ❖ No .gitignore: will come later
- ❖ License file: None – or GNU?
- ❖ Project Lead shall add collaborators („+“ Menu)
- ❖ e-mail: follow link and accept invitation

A Short Tour through GIT

- ❖ Install SourceTree (<https://www.sourcetreeapp.com/>)
- ❖ Collaborator: SourceTree->Neu / Klonen have to clone the repository (SourceTree Remote)
 - ❖ Klick on Globe on the right side of „Source Pfad / URL“
 - ❖ Klick on „Clone“ and the repository will be copied to your local drive.

Round Trip I

- ❖ PL creates in local Repository a simple Text File
- ❖ Stage, Commit and Push on Repository
 - ❖ Stage: Collect Files for Commit
 - ❖ Commit: Commit Files to local Repository
 - ❖ Push: Upload File to Repository at e.g. www.github.com
- ❖ Collaborator pull this file from Repository
 - ❖ Pull: Copy File from Web Repository to local Repository

Round Trip II

- ❖ Collaborator shall create a new Text File:
 - ❖ Stage, Commit, Push
 - ❖ PL shall pull new File from Web Repository
- ❖ PL and Collaborator edit same File:
 - ❖ Collaborator pushes File first
 - ❖ PL tries to push the File afterwards
 - ❖ GitHub refuses push

Round Trip III

- ❖ PL has to solve Merge Conflict
 - ❖ Use Mine / Theirs or Merge manually
 - ❖ After Merge Commit and Push is allowed
- ❖ Collaborator to pull merged version

Exercise I

- ❖ Create a test Repository on GitHub.com
 - ❖ Create accounts for PL and collaborator
- ❖ Do the Round Trip
 - ❖ PL: Create, stage, commit, push of simple text file
 - ❖ Collaborator: Create, stage, commit, push of simple text file
 - ❖ PL & Collaborator: Pull Files of your counterpart

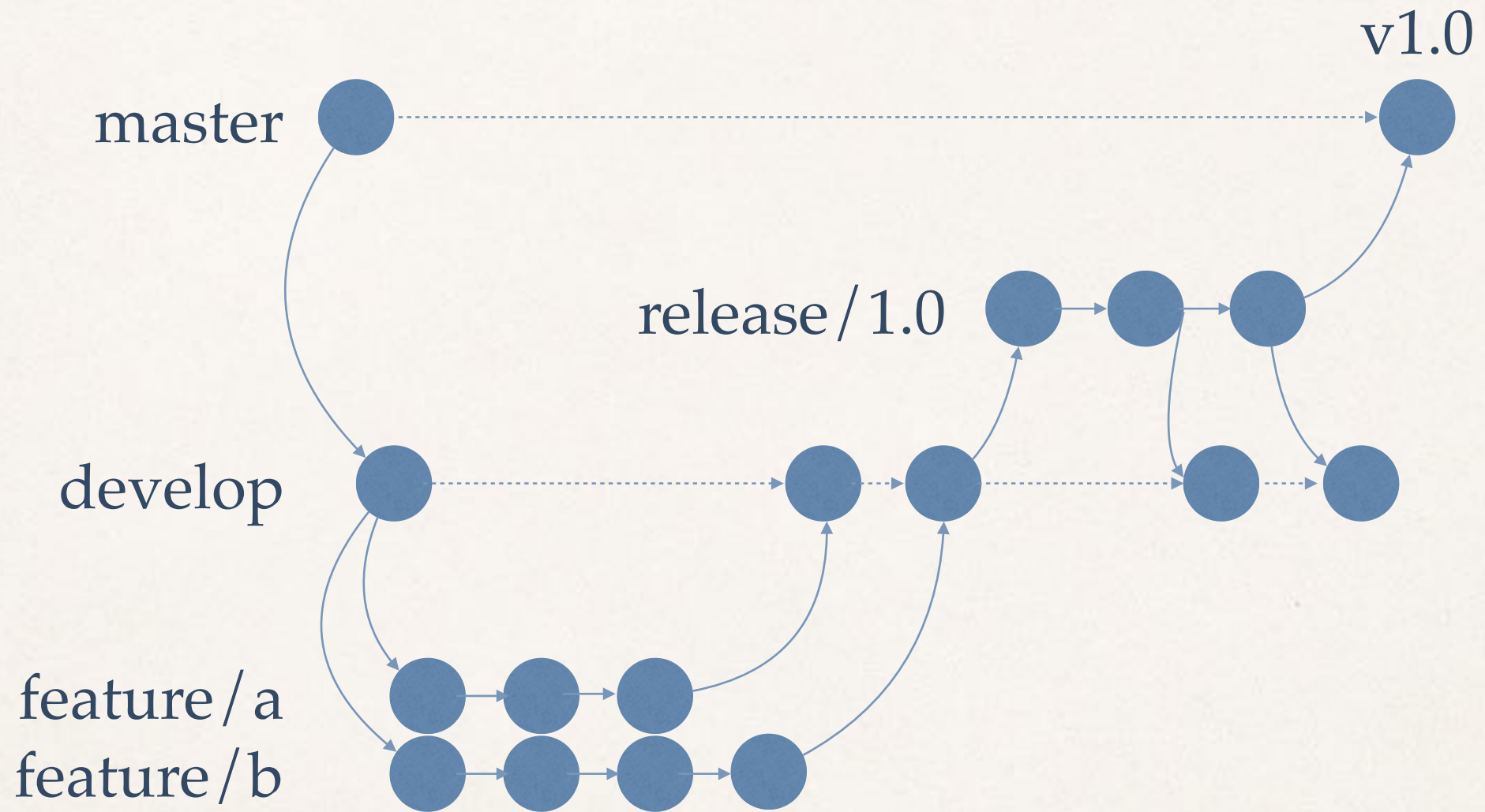
Exercise II

- ❖ PL and Collaborator edit same file
 - ❖ Collaborator pushes first
 - ❖ PL has to solve merge Conflict
 - ❖ Collaborator has to pull merged version

.gitignore

- ❖ Build artifacts which should be ignored
 - ❖ .o, .war, .jar, .exe, .mdb, etc.
- ❖ .gitignore is in the root directory of your project
- ❖ <https://www.gitignore.io> helps to set up
 - ❖ Input: Used OS, used IDEs, Tools, etc.

Branching in git



<https://blog.sourcetreeapp.com/2012/08/01/smart-branching-with-sourcetree-and-git-flow/>

Assignment

- ❖ Create a configuration management plan for your project
- ❖ It should contain at least
 - ❖ Versioning methods & tools
 - ❖ Which tools do you use?
svn, git
- ❖ Do you have all documents under version control?
- ❖ If not, why not and how do you monitor changes?
- ❖ Organization of artifacts
- ❖ Variant management