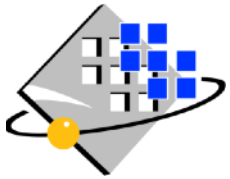
université
PARIS-SACLAY



L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

Part IV : Version and Configuration Management

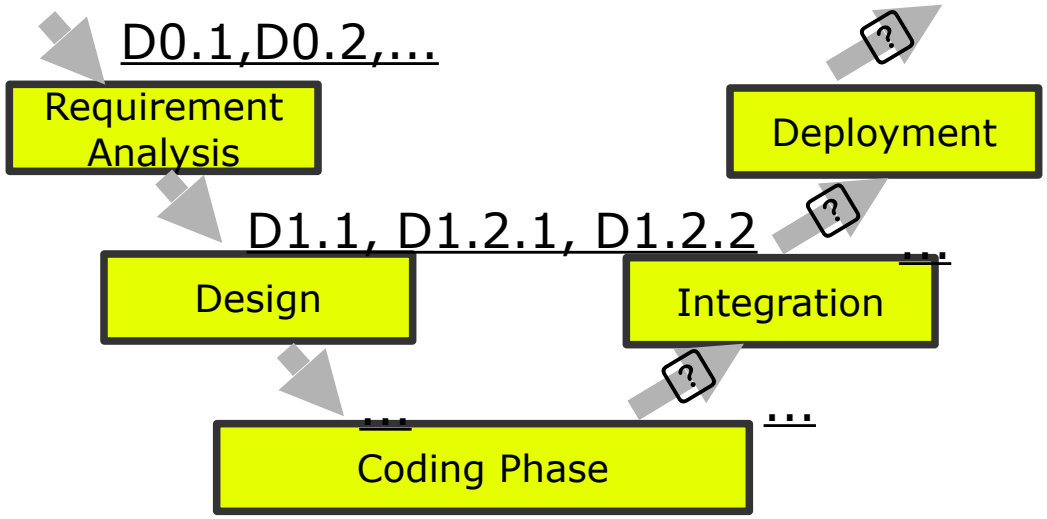
Burkhart Wolff
(burkhart.wolff@universite-paris-saclay.fr)
https://usr.lmf.cnrs.fr/~wolff

Plan of the Chapter

- Motivation: Version and Configuration Management as „the" means for collaborative development
- Version management
  - Centralized Version Control
  - Distributed Version Control
  - Organizing Merges
- Beyond Versions: Configurations
  - Build Management
  - Advanced Configuration Management

Motivation

- Recall: SE Processes are based
  - on a large flow of documents and code
  - … that have to be edited collaboratively
  - … distributed consistently
  - … while controlling access



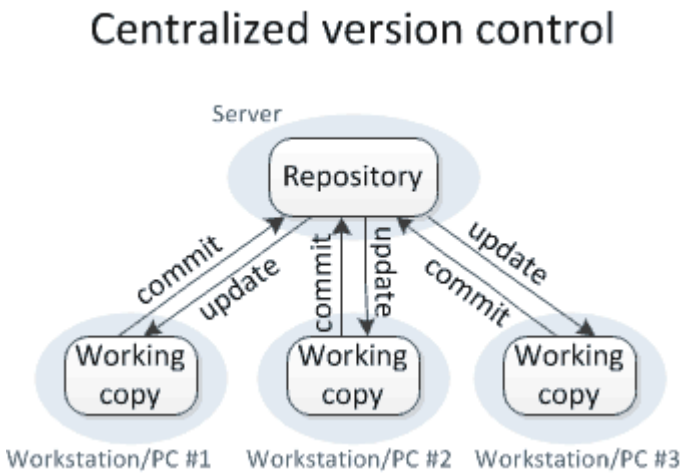
Motivation

- Important TECHNICAL means to support the process
  - Explicit tools for Version Management

    (CVS, svn, git, mercurial, sourcedepot, ...)
  - Create and track revisions of files and file-trees
  - Create and track differences between files and file-trees
  - Access control to the various parties of process
  - Locking of documents
  - Merging of revisions
  - Quality control
  - . . . actually, it is dead-useful for everything !!!

Concepts of Centralized Version Control

- Working copises
  (in user space)
- Repository
  (on the server-side)
- update:
  syncing with the
  repository



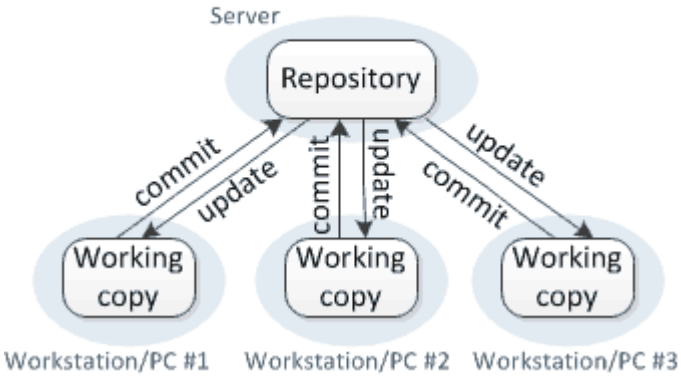
- commit: creating a new revision of a document
  (involves new registration, inclusion in documents,
  consistency checks)
- operations lock, checkout, import, ...

Concepts of Centralized Version Control

- First widely used system: CVS
- Nowadays in use : svn
- In connection with a gui-client: useable for end-users ... ... for everything ...
-

https://subversion.apache.org/

SUBVERSION®



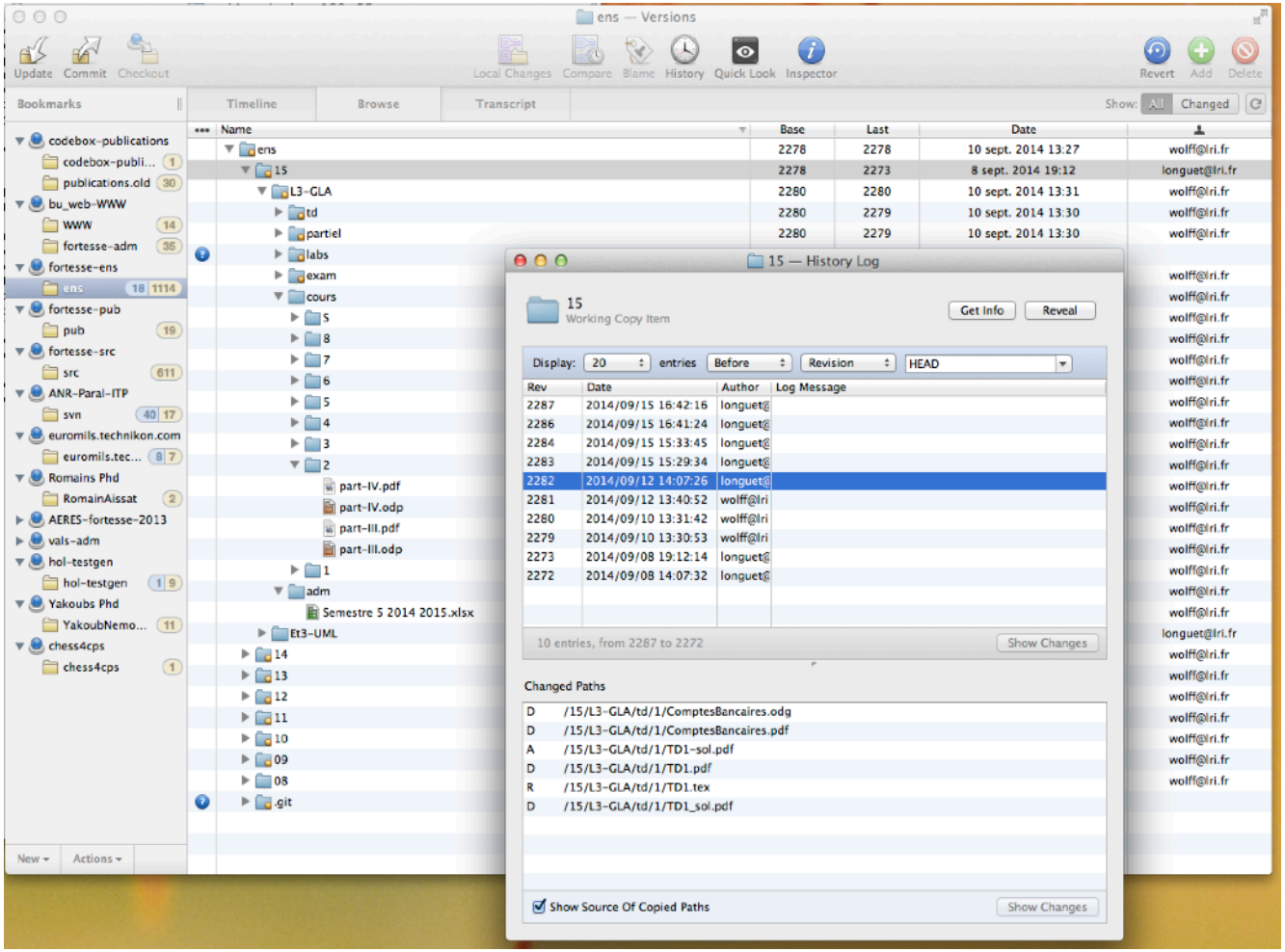
… for everything …

- my working copy for this course material

…

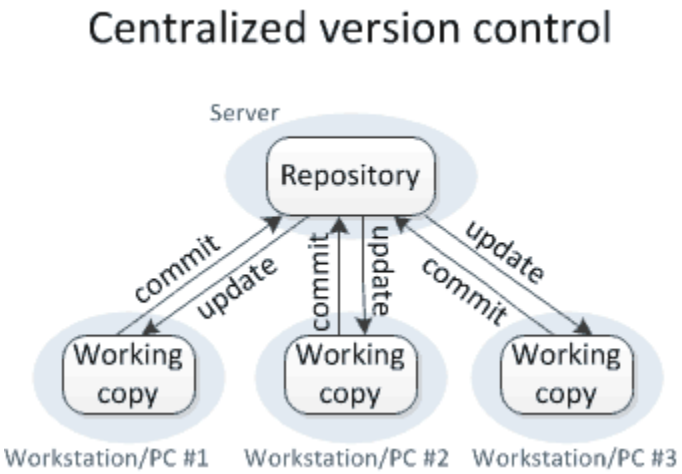
- Version - management is not just for code

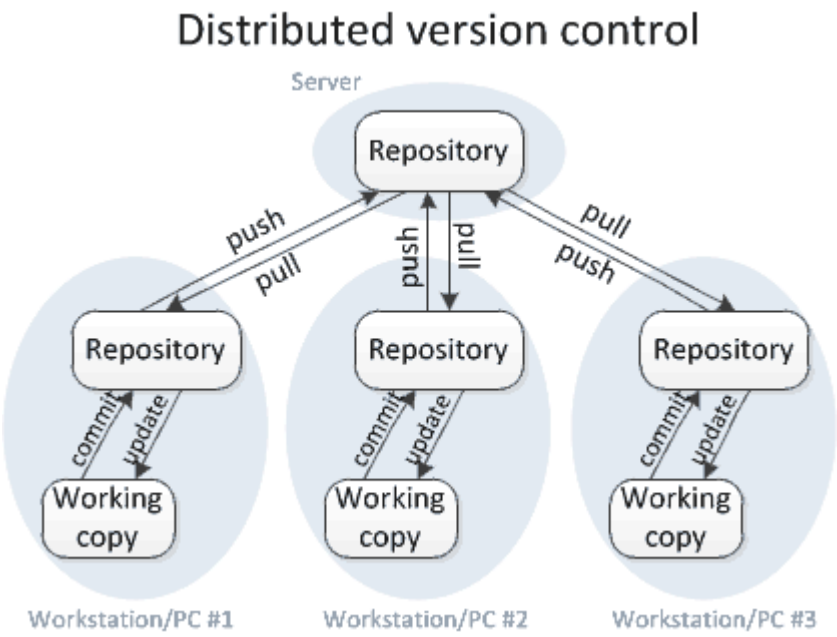
…



One step further:
Distributed Version Control

- Hierarchy of repositories:



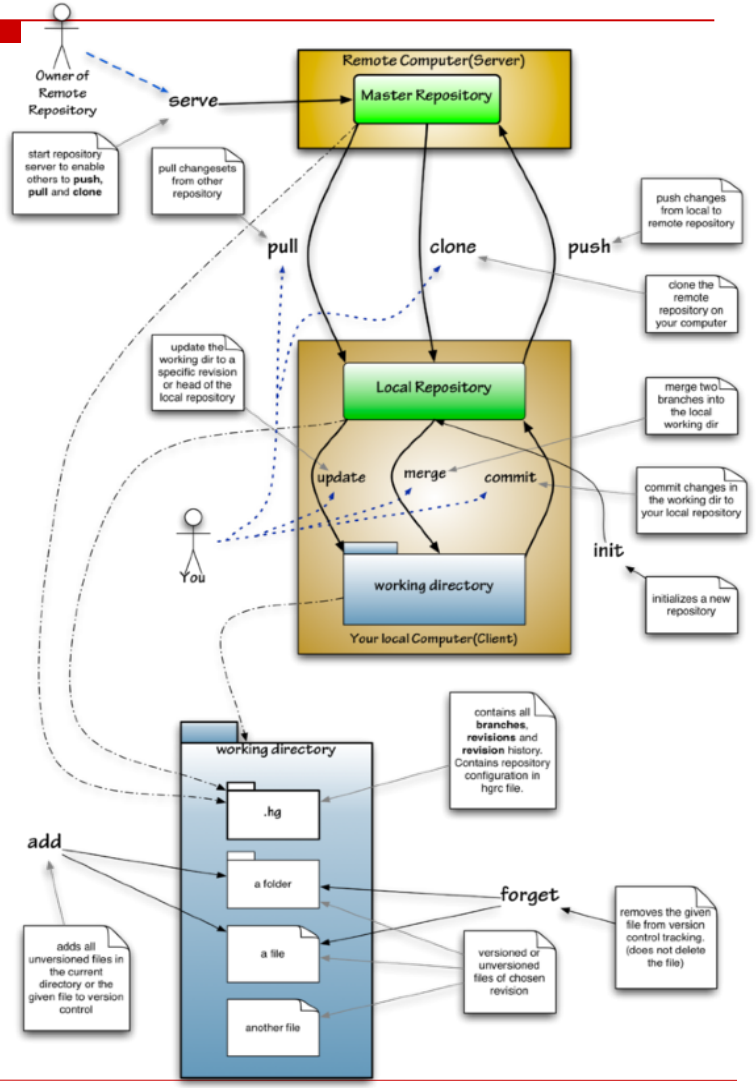


- one more sync-level, but more precise history in practice… since everybody can check in locally
- hierarchy strictly speaking not necessary

Distributed Version Control Systems

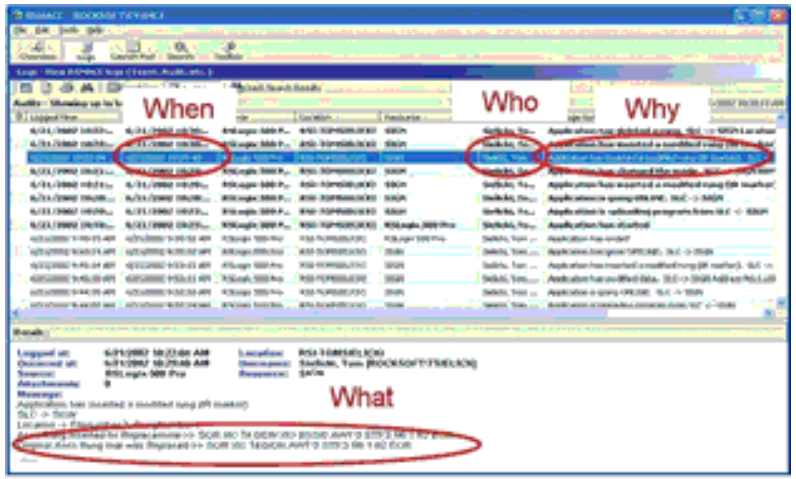
- Opensource:
  - git (Linux)

  - mercurial (Isabelle)
    http://mercurial.selenic.com/



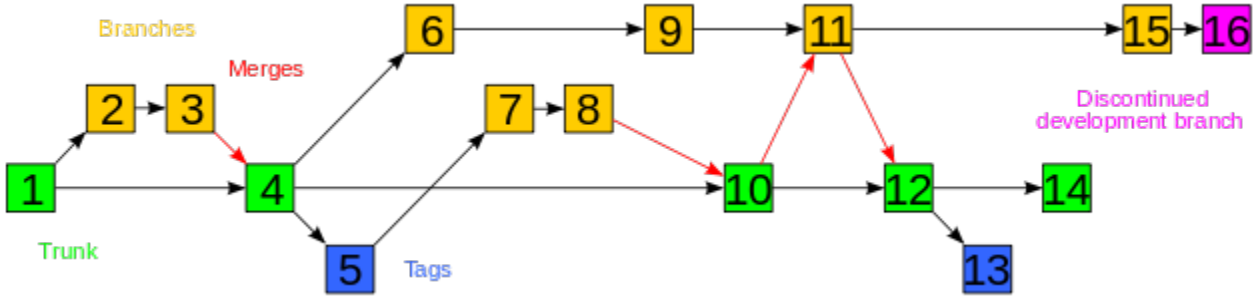
Consequences for a development process

- The entire development becomes visible, trackable, reproducible, and can be an object in its own right (if formal): apply changeset XYZ … transfer devel from repo A branch XX to repo B on branch YY …



Problems of a distributed development

- If not synced via explicit locks,
  a development looks like this can be organised via "merges"

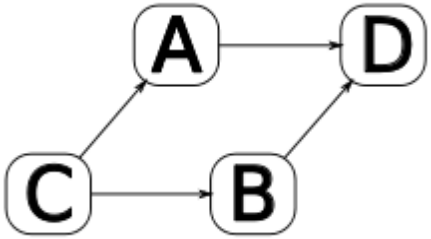


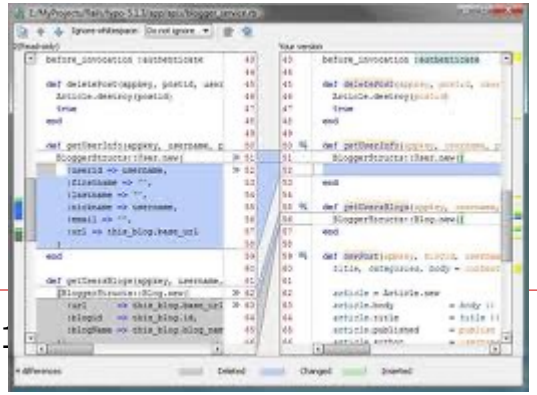
- Preference for re-generation binary formats
- Merging: For binary formats, only special purpose merges work (as in MS Word, for example ...)
- For textual formats :

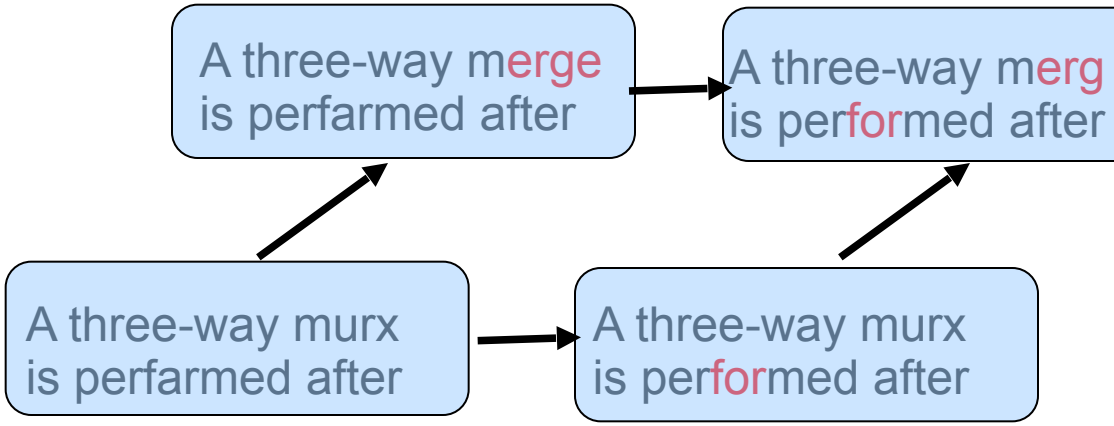
A partial solution ...

- For textual formats:

  A three-way merge is performed after an automated difference analysis between a file 'A' and a file 'B' while also considering the origin, or common ancestor, of both files. It is a rough merging method, but widely applicable since it only requires one common ancestor to reconstruct the changes that are to be merged.



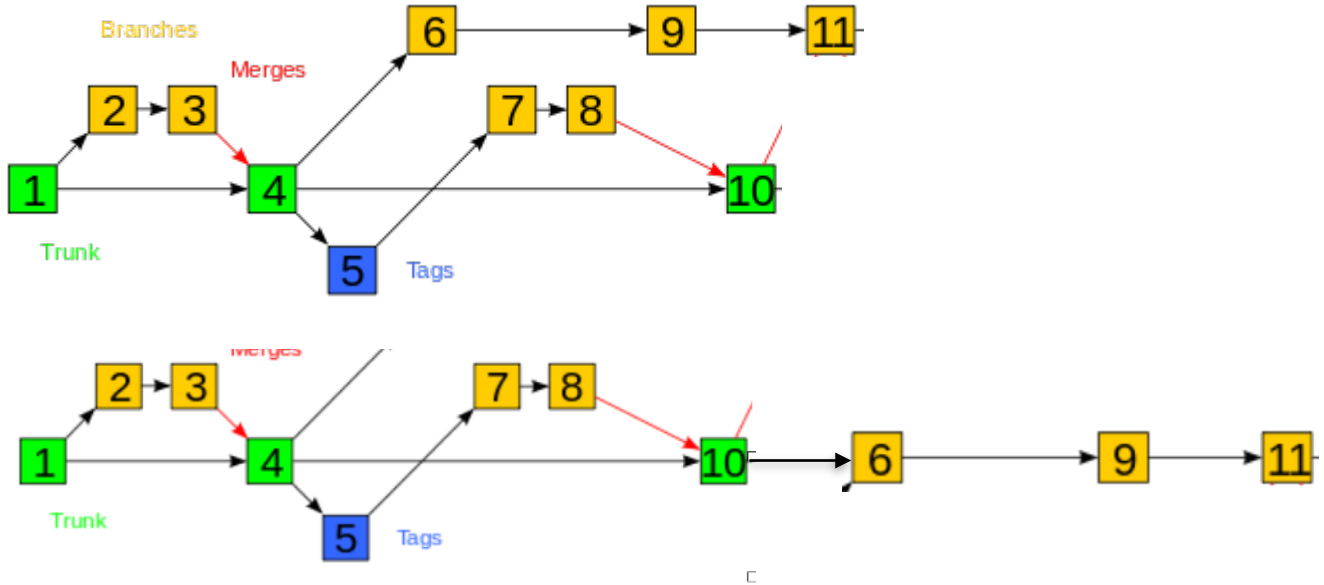




Tools: For example xmerge, which also offer conflict resolution by hand ...

Problems of a distributed development

- Another strategy of organising developments are "rebases":



- Preference for re-generation binary formats

A better solution ...

- In a distributed development process, where a very large number of merges occurs routinely, the validation of the intermediate results becomes crucial.

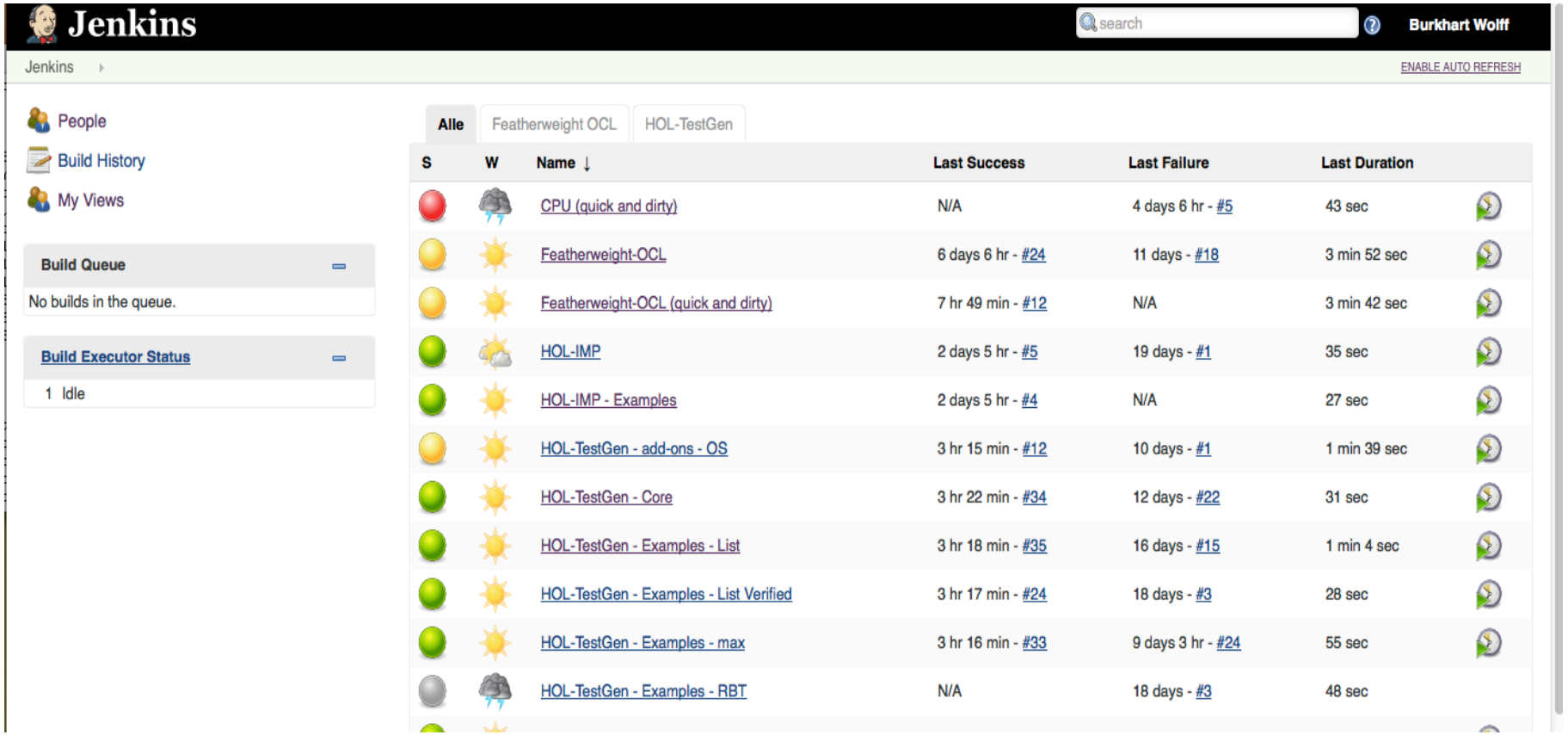
  (This limits the use of informal documents.)

  - validation on any check-in
    (for example: automated type-checking)
  - validation for UML - documents
    (self-defined consistency checkers for UML models)
  - automated static analysis and tests during systematic and periodic builds ...
  - ... more advanced methods, using proof techniques.

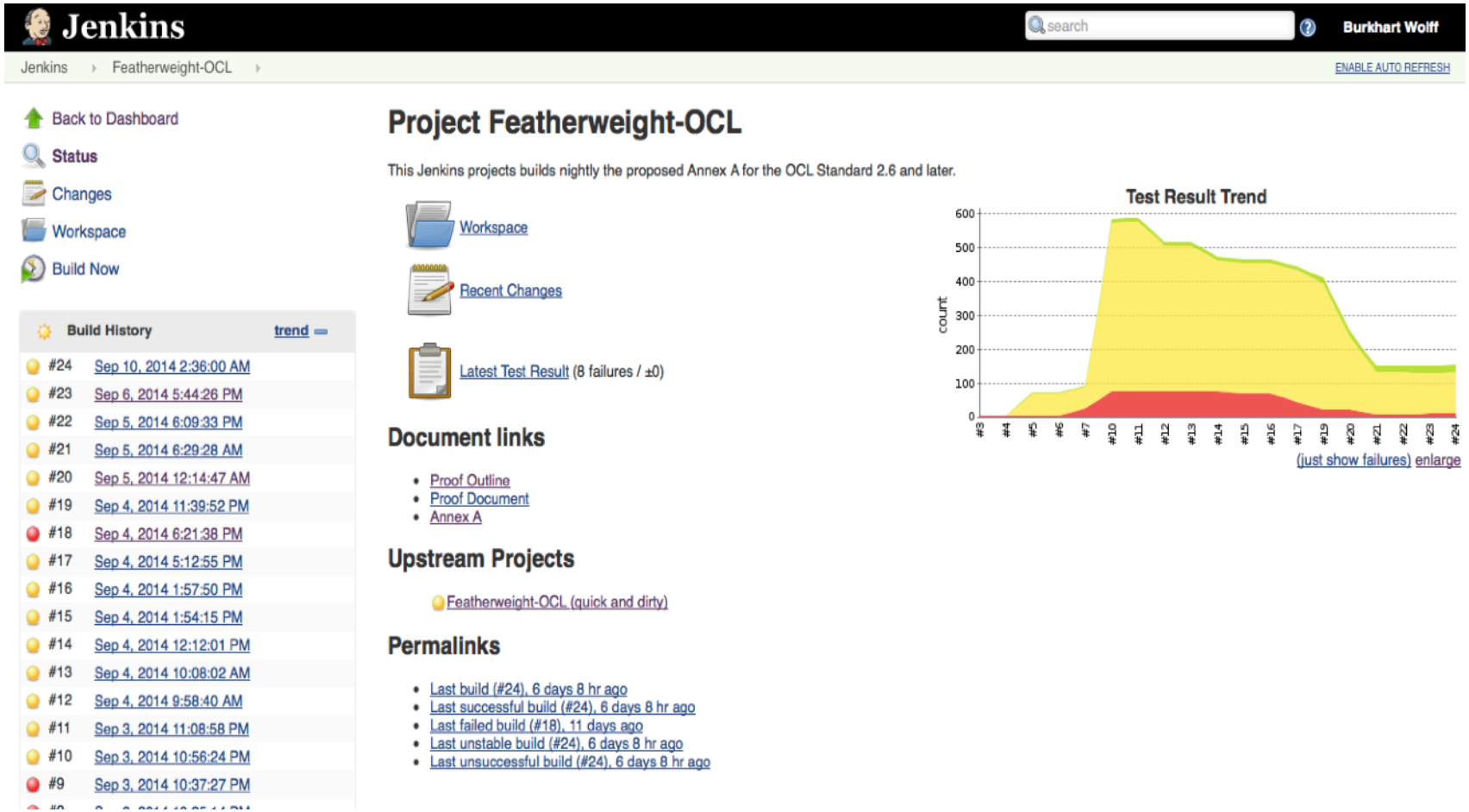
Build Management: A Build-Server

- Pervasive Version-Management allows for a "continuous build / continuous integration" element in the dev-process
  - Update all sources from the repository
  - Re-compile everything
  - Rerun all static checks (lint, code analysis…)
  - Rerun all executable test-suites
- Open Source Solution : Jenkins

Build Management: A Build-Server



Build Management: A Build-Server



Towards Configuration Management

- ... generalizes Version-Management
  - by configuration descriptions (including functionality environment, hardware,...)
  - attempts to GENERATE a revision on the basis of meta-data over dependencies and change-sets
  - works with heavy virtualization techniques nowadays (Google, Microsoft)