Software Architecture

Bertrand Meyer, Michela Pedroni

ETH Zurich, February-May 2010

Lecture 11: Configuration management
About your future

- You will never work alone.
- Other people will mess up your code.
- You will mess up other people’s code.
- You will never write a (major) program from scratch.
- The software you will work on was already there when you joined the company.
- The software you will work on will still be there when you leave the company.
Configuration management – the long story

“Configuration management is unique identification, controlled storage, change control, and status reporting of selected intermediate work products, product components, and product during the life of a system.”

(Anne Mette Jonassen Hass: “Configuration Management; Principles and Practice”)
Configuration management is about the role of **TIME** in software development.

It is the task of tracking and controlling changes.
Ten key SCM activities

1. Accessing and retrieving software
2. Retrofitting changes across the development life cycle
3. Migrating changes across the development life cycle
4. Managing the compile and build process
5. Managing the distribution of changes
6. Obtaining approvals and sign-offs
7. Managing software change requests
8. Coordinating communication between groups
9. Obtaining project status
10. Tracking bugs and fixes

(Software Configuration Management, Jessica Keyes)
Change management

- CM has to record
  - WHICH document was changed.
  - WHAT was changed.
  - WHO has done the change.
  - WHEN was the change made.

- The history of the changes should be made visible.
- It should be possible to undo all changes.
- It should be possible to view the version of a document at a certain point in time.
Three parts of SCM

1. Version control systems
   a) Local version control
   b) Central version control
   c) Distributed version control

2. Build management systems

3. Bug and issue tracking systems
Three parts of SCM

1. Version control systems
   a) Local version control
   b) Central version control
   c) Distributed version control

2. Build management systems

3. Bug and issue tracking systems
Versions (also called revisions) give a unique time-dependent identification to each document.

Deciding for proper version names is the basis for successful software configuration management.

There can be multiple levels of versioning schemes within one project. Examples:

- Versions for written documents:
  - Example: REQDOC-20100103-R4

- Versions for source code:
  - 1.1, 1.2, 1.3, 1.4, 1.4.1, ...

- Versions for binary builds:
  - 610, 611, 612, ...
A system that stores and organizes documents over space is called a **file system**.
A system that stores and organizes documents over space and time is called a repository.
Version control systems (VCS) manage...

- Source code revisions and variants
- Binary versions of the software (builds)
- Requirements and analysis documents
- Design documents, UML diagrams
- Test results
- System configuration information
- ...
VCSs are about...

- ... knowing who has done what to which document in the past.
- ... enabling different people at different locations to work on the same set of documents at the same time.
- ... going back to an old version of the documents in the case that the path taken was not good.
- ... tracking the quality of the software over time and to stop software regression.
Local version control

- Only one document under version control
- Versions of the document are stored in the document itself
- Examples:
  - 1972: First version control software called SCCS (Source Code Control System), Bell Labs
    - Delivered with Unix
  - Early 1980s: RCS (Revision Control System), Purdue University
    - Part of GNU
    - Automates storing, retrieval, logging and identification of revisions
    - Stores deltas
    - Uses locking of files
Lock modify write

- Also called “lock modify unlock”
- File is locked by a user for modification
- System prohibits changes by other users when locked
- Binary files need such an approach...

Advantage: No conflicts
Drawback: Wait until file is unlocked

Based on the philosophy of pessimistic version management
Demo RCS

1. Create a file
2. Check-in: `ci test.txt` → creates `test.txt,v`
3. Check-out: `co test.txt` → generates `test.txt`
4. Edit `test.txt`
5. Check-in changes: `ci test.txt` → But we need a lock!
6. Retroactively lock file: `rcs -l test.txt`
7. Have a look at `test.txt,v`
Central version control

- Client server system with access control
  - Server stores versions of a project and history
  - Clients check out a local working copy

Examples:
- 1986: CVS ( Concurrent Versions System )
  - Uses RCS internally
  - Per default no locking (but can be enabled)
- Early 2000s: SVN ( Subversion )
  - “A better CVS”
CVS

- Best known SCM tool.
  [http://www.cvshome.org](http://www.cvshome.org)
- Key to most open-source development projects and used in many companies.
- Based on RCS (Revision Control System)
- Command-line program (!)
- Graphical-User-Interfaces:
  - WinCVS (Windows) / GCVS (Unix)
  - Integration into Eclipse and many other IDEs
CVS (cont.)

Features of CVS:

- Central repository
- Checkout creates a copy of the files on the local machine.
- File-based versioning
- Distributed work over the Internet with PSERVER (insecure) or SSH (secure) protocols.
- Can work with binary and ASCII encoded files, but is not very effective with binary files.
- Can do some primitive conflict resolving.
Views

File A
1.0 1.1 1.2 1.3

File B
1.0 1.1 1.2

File C
1.0 1.1 1.2 1.3 1.4 1.5

View at time T

Time
Head = View at time $\infty$
Tagging/labeling

Tag/Label = Important snapshot in time
Branching

Creating multiple variants of a set of documents is called branching (or forking)

1.0 1.1 1.2 1.3

1.1.2.1 1.1.2.2 1.1.2.3

Trunk (sometimes called baseline or mainline)

Branch
Merging

Joining variants that were developed independently for some time is called merging.
Distributed development

**checkout:** create local working copy of repository

**commit:** write your changes to the repository

**update:** merge changes from the repository into the local working copy
Distributed development

resolve: user intervention to address a conflict between changes on the same document
Copy modify merge

- Allows simultaneous modifications of a document
- Changes need to be merged automatically or manually

Advantage: Simultaneous editing
Disadvantage: Not applicable to binary files

Based on the philosophy of optimistic version management

Used by all modern centralized and distributed version control systems
What is wrong with CVS?

- File based management
  - Rename file
  - Changing directory structure
- No atomic commits
- Network protocols are problematic
  - PSERVER is not encrypted
  - RSH is obsolete
  - SSH lacks anonymous access
  - Too much communication between client and server
- Inefficient storage of binary files
Subversion (also known as SVN)

Available at [http://subversion.tigris.org/](http://subversion.tigris.org/)
Full book on Subversion available at [http://svnbook.red-bean.com](http://svnbook.red-bean.com)

“Subversion is meant to be a better CVS, so it has most of CVS's features. Generally, Subversion's interface to a particular feature is similar to CVS's, except where there's a compelling reason to do otherwise.”

Subversion Homepage
Subversion uses a client server architecture
Every developer works on his/her “working copy”
Version numbering SVN

Version numbering per “commit”

File A

File B

File C

Head = View at time $\infty$
HEAD, BASE, COMMITTED, and PREV

File A

HEAD = revision 9
BASE = revision 7
COMMITTED = revision 6
PREV = revision 5

Working copy (revision 7)
Directories and changes

- SVN tracks tree structures, not just file contents
- You may move files (move = copy + delete)
- You may add/delete/copy/move directories, but the changes in the repository only take place after committing
- Directories have version numbers
- Branching means copying the directory structure into a new place in the repository (using a cheap copy)
Subversion Commands

- `svn checkout`
- `svn update` / `svn revert`
- `svn commit`
- `svn info` / `svn log` / `svn status`
- `svn add` / `svn delete` / `svn move` / `svn mkdir`
- `svn copy`
- `svn diff`
- `svn merge` / `svn resolved`
- `svn cat` / `svn list` / `svn blame`
- `svn export` / `svn import` / `svn switch`
Distributed version control systems (DVCS)

- Every working copy is a repository with version history (and thus a backup of the code base)
- Synchronization uses the exchange of patches (change-sets) with unique ids between peers
  - “Pull” changes from other repositories
  - “Push” changes to other repositories
- Usually one copy is sanctioned as main development branch
- Examples:
  - 2005: Git, Linus Torvalds
  - 2005: Mercurial
Centralized VCS

Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/
Workflow centralized VCS

http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/
Operations

Bootstrap
- init
- checkout
- switch branch

Modify
- add, delete, rename
- commit

Information
- status
- diff
- log

Reference
- tag
- branch

Decentralized
- clone
- pull, fetch
- push
Distributed VCS

Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/
Workflow distributed VCS

http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/
Objects in Git

File types:
- Blobs
- Tree
- Commits
- Tags

Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/
Objects in Git

File types:

- Blobs
- Tree
- Commits
- Tags

Git the basics, Bart Trojanowski,
http://excess.org/article/2008/07/ogre-git-tutorial/
Objects in Git

File types:
- Blobs
- Tree
- Commits
- Tags

Git the basics, Bart Trojanowski,
http://excess.org/article/2008/07/ogre-git-tutorial/
Git example - bootstrapping

$ mkdir project
$ cd project
$ git init

Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/
Git example - bootstrapping

$ echo test > test

Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/
Git example - stage

$ echo test > test
$ git add test

Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/
Git example - commit

$ echo test > test
$ git add test

$ git commit -m "test"
Created initial commit 6f01040: test
1 files changed, 1 insertions(+),
0 deletions(-)

create mode 100644 test

Git the basics, Bart Trojanowski, http://excess.org/article/2008/07/ogre-git-tutorial/
Remote repositories

- Use "git clone" + remote url to get a local copy
- Use "git pull" to fetch and merge remote changes
- Use "git push" to publish local changes to remote repo

- Always pull before push
Advantages/disadvantages of DVCS

Advantages:
- Everyone has a local sandbox
- Works offline
- Fast diffs, commits, reverts
- Handles changes well
- Easy branching and merging
- Less management

Disadvantages:
- Not a backup
- No real "latest version"
- No real revision numbers
Binary builds

There are different levels of binary builds:

- Daily (Nightly) Builds
- Integration Builds
- Stable Builds
- Releases
  - Alpha Release
  - Beta Release
  - Release Candidate
  - Official Release

current, but not reliable

not current, but reliable
Bug tracking

Bug-Tracking is the term for an infrastructure that captures and manages bug-reports in a given project.

The number and quality of bugs is normally a judgment for the release-quality of software.
CM Terminology

- Repository, Commit, Update, Checkout, Head
- Branch/Fork, Merge
- Conflicts, Resolving Conflicts
- Tag, Label
- Pull, push, fetch
- Release, Build
- Test Cases, Test Suite, Regression Tests
Tools for CM

Software Configuration Management (SCM) Tools
- RCS, CVS, Subversion, Git, Mercurial, Monotone, ArK, tla (FOSS)
- ClearCase, BitKeeper, SourceSafe (Commercial)

Bug-Tracking
- Bugzilla (FOSS)
- Origo issue tracker

Daily Build
- Xenofarm (FOSS)
- Tinderbox (FOSS)
- Most projects have customized software