

SOFTWARE CONFIGURATION MANAGEMENT

Cape Town SPIN - Albert Visagie - 19 March 2008

Goals

- Where were we?
And get back there reliably.
- Where are we?
How did we get here? Which bugs were fixed in this version? who did it? Was it properly tested? Do we have everything?
- Share work efficiently.
Minimize toe-treading. Consistency. Reliability. Traceability.

Overview

- Stories
- Theory, best practices
- Recommendations
- Tools
- Resources

Horror Stories

- The tarball that came flying over the wall.
 - 2 longer term people, many students, one professor.
 - 6 months' changes, 300k lines of scientific code, customized.
 - New product code.
 - Linux, new build must target Windows.
- “I did not have the time to look for bugs introduced by other people.”
Implications!

Horror Stories

The case of the patches...

□ Growing company and product:

- 9 developers on server-side. Also hardware, firmware & embedded OS.
- 4-5 releases running in the field, world-wide.
- Only two versions: stable and development – in Visual SourceSafe using exclusive locking.
- Binary patches.
- Very competent, disciplined people.

□ Pain points

- Releases live outside the versioning system because it is too slow.
- Binaries running at customer site are not linked to our release process.
- Patches made by “oracles”,
- Test team frustrated because it very difficult to know what is in a build – errors are frequent.

Other danger signs

- “This is the Code Freeze, no-one may check code in until we release.”
- “Send me a copy of your files, so that I can make mine work with it...”
- “Mine works, are you sure you have the right files?”
- “We use this tool in development, but builds are done that way, be sure check that it works!”

SCM Best Practices

Underlying purpose:

Manage divergence specifically

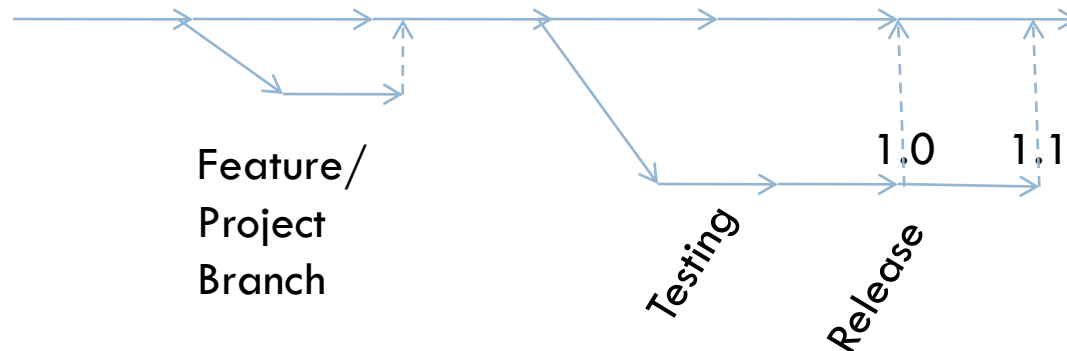
References:

- ▣ “High level best practices in Software Configuration Management” – Wingerd & Seiwald, <http://www.perforce.com>
- ▣ Software Configuration Management Patterns - Berczuk & Appleton. (<http://www.scmpatterns.com/>)

SCM Best Practices

□ Codelines

- ▣ Policy for each codeline (stability, tests passing, build works)
- ▣ Main-line for going forward.
- ▣ Branch by purpose, not by version.
- ▣ Changes from feature lines or release lines must be considered for propagation back to the main line.



SCM Best Practices

- Workspaces
 - ▣ Like a desk...
 - ▣ Never share code outside the SCM system
 - ▣ Stay in sync! (manage divergence)
- Branches: Creating new codelines
 - ▣ Changing, incompatible code line policy
 - ▣ Branch late
 - ▣ Branch instead of freeze
- Change propagation
 - ▣ Make original changes in the branch that evolved the least
 - ▣ Propagate early and often

SCM Best Practices

□ Builds

- Everything that is needed to make the build, exactly
- Standardize tools and environment
- Place output separate from source
- Build often
- The build happens from a line that has a policy of being more mature, and buildable!

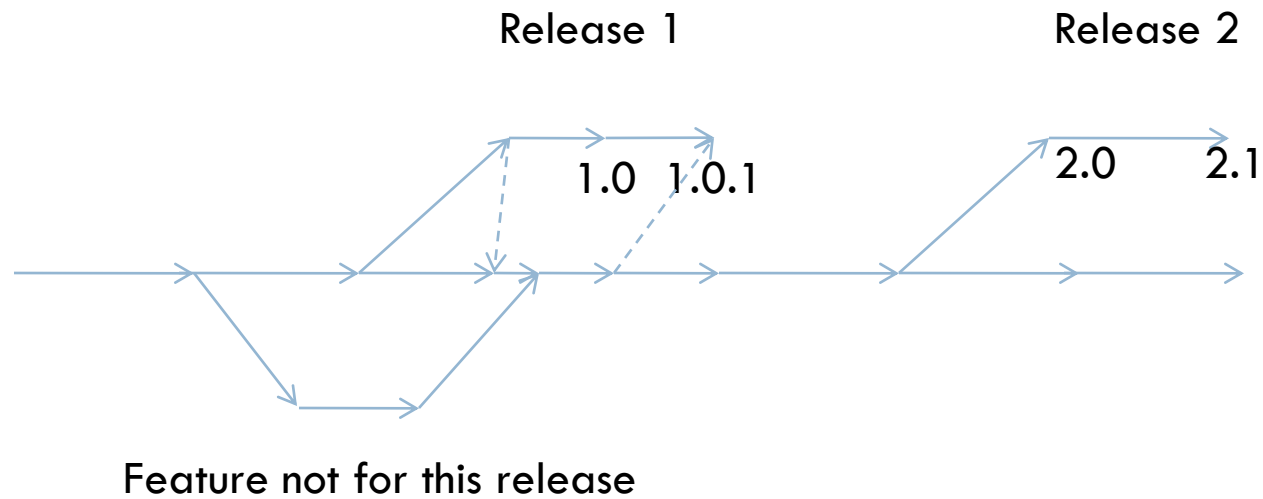
□ Process

- Track change packages. (linked to requirements, change requests)
- Track propagation of change packages
- Easy access to process documents and references

Lone Developer to Small Teams

- Structure (in order of relevance as people join, and more customers come)
 - ▣ Mainline for all work
 - ▣ Branch for releases, tag builds
 - ▣ Branch for major features, bugs or experiments
 - ▣ Propagate applicable changes from releases back into the mainline
- Benefits:
 - ▣ Improved manageability, even for one person
 - ▣ Better chances of convergence in features and stability
 - ▣ Less dependence on one person for builds and fixes
 - ▣ Effortless backup with history
 - ▣ Consistency when sharing code
 - ▣ Great tool support, free ones too

Lone Developer to Small Teams



Web Development

For small groups, features:

- ▣ Rapid changes
- ▣ Changes touch various parts of the system: database, templates, logic, and different people's work is more prone to affect each other.
- ▣ Primarily as a code sharing tool, tracking sequential changes & managing divergence:
 - Update often
 - Commit changes often

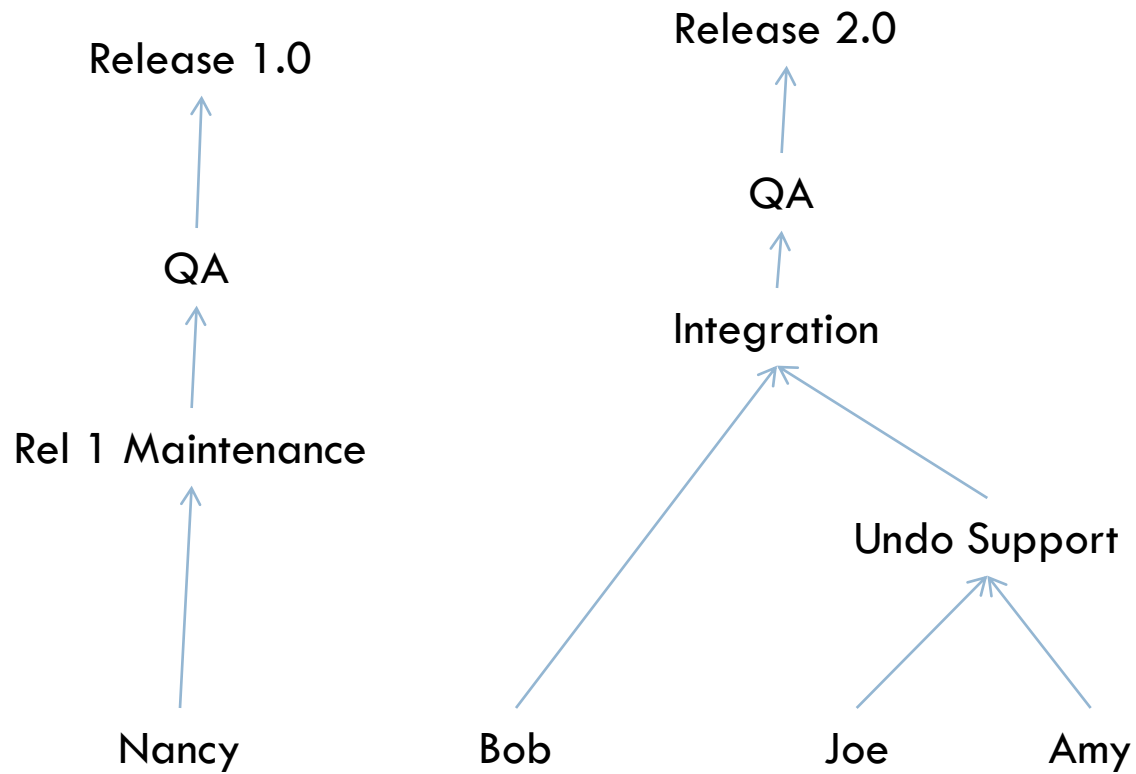
Larger Teams

Split the codeline into multiple codelines with controlled propagation of changes between them:

- *Private Workspaces* for developers
- *Development lines* for integration of specific projects or small teams
- The *integration line* collects changes from the development lines.
- *QA line* for collecting changes from the integration line once it is ready.
- *Release branch* for each release.

Changes linked to issue tracking

Larger Teams



Tools: How does it work?

- Locking
 - ▣ Shared files
 - ▣ Lock it to modify it
 - ▣ Workable for small teams, largely decoupled from each other
 - ▣ Some types of content require locking: Word docs, Images, binaries . . .

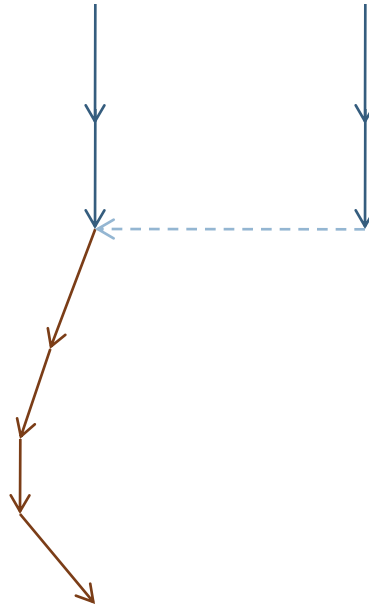
Tools: How does it work?

- Copy-modify-merge
 - Centralised
 - Edit concurrently in local working copies
 - Update to merge latest changes from server in
 - Commit changes to append to latest in server

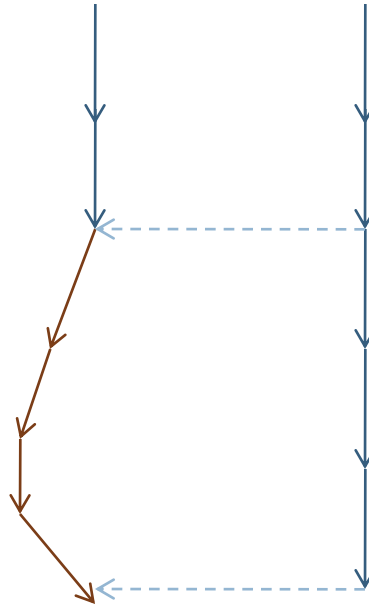
Tools: How does it work?

- Directed Acyclic Graphs
 - ▣ A version is a directed acyclic graph of changesets
 - ▣ Changesets have 1 or more parents (more means merged lines)
 - ▣ Offline operation
 - ▣ Easy to build the idea of code lines using such graphs.
 - ▣ Modern distributed SCM systems rely on the DAG methodology.

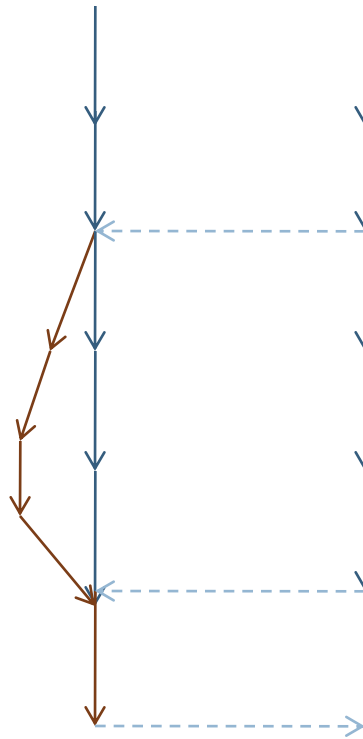
Directed Graphs Example



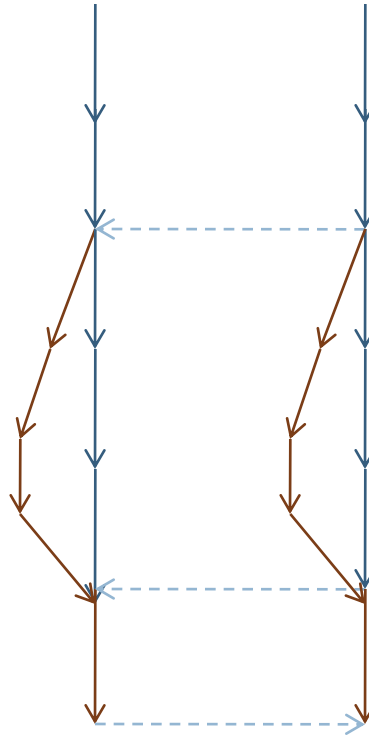
Directed Graphs Example



Directed Graphs Example



Directed Graphs Example



Questions about change set driven tools

- Can it groups changes into changesets?
- Can one cherrypick i.e. Only pick out certain changes or even change sets?
- Not sure about the free ones
- Perforce warrants investigation

Features to consider

- ❑ Moving/renaming
- ❑ Triggers (scripts)
- ❑ Atomic check-ins/commits
- ❑ 3-way merge (base, this, other)
- ❑ Distributed vs. Client Server
- ❑ Streams or Promotion levels
- ❑ Issue tracking integration
- ❑ How does it handle binary files?

Tools

- Distributed:
Generally all DAG based.
 - ▣ GIT (Linux Kernel, no windows client)
 - ▣ Mercurial (Netbeans recently switched to it)
 - ▣ Bazaar (Ubuntu)
- Client Server
 - ▣ Subversion (Please don't even think about using CVS)
 - Bad rename support, no merge tracking (use 3rd party, tbd in 1.5)
 - Good support in Eclipse and GUIs
 - Trac
 - Fisheye
 - SmartSVN (client)
 - TortoiseSVN (client)

Tools

□ Commercial

▣ Accurev

- Streams methodology outlined in “Larger Teams”
- Heard very good reports
- Stream browser explicitly links how changes propagate between streams

▣ Perforce

- Change propagation methodology very close to the DAG model makes the *Main Line* idea easy and safe through good intuitive change propagation.

▣ Microsoft:

- Visual SourceSafe: A clunky shared folder with some merging support.
- TeamSystem: Not sure, reported to be good, but it is a whole project management and collaboration suite, tight with Visual Studio.

Closing

- Simple process
- Write it down
- The test is whether a new developer can reliably reproduce old builds and fix simple bugs.
- Remember the Goals . . .

Resources

- Mostly vendor neutral whitepapers
 - <http://www.accurev.com/software-configuration-management-resources.htm>

Specifics:

 - http://www.accurev.com/wp_ascade/
 - <http://www.perforce.com/perforce/technical.html#whitepapers>

Specifics:

 - <http://www.perforce.com/perforce/bestpractices.pdf>
- <http://www.scmpatterns.com/>
- Distributed systems:
 - <http://www.selenic.com/mercurial/wiki/>
 - <http://bazaar-vcs.org/>
 - <http://git.or.cz/>