Version control and collaborative working

- We will consider:
  - Version control
  - Collaborative working
  - Tools: CVS, Subversion (Subclipse), Git, …

Version management/version control/releases

- Consider system versions:
- The configuration database contains information about system versions/configurations
  - As particular selections of particular versions of components
  - A system version is any instance of the system that differs in some way from others:
    bug fixes, restructuring, new features…
  - A *system release* is a version of the system that is distributed to clients
- There are usually many more *versions* of a system than *releases*
  - Internal development or testing versions
- The details of a system release should be recorded
  - In case of a need to rebuild in the future

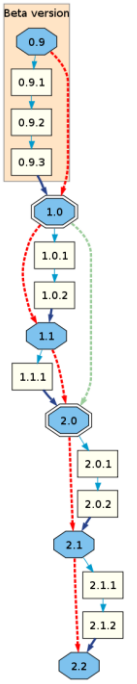
- Consider component versions:
- The configuration management database *identifies* particular version of components in some way
  - But it *does not contain* the components themselves
- The actual components are managed by a version management or version control system (VCS)
  - This may be integrated into the configuration management database
  - Or a separate system, more or less loosely coupled to the configuration management database
- The VCS must implement a unique way of referring to the different versions of components (which do not change their name):
  - The most common way is a hierarchical version numbering scheme

Version numbering

- Many schemes are based on a major.minor.patch approach (e.g. see Semantic Versioning http://semver.org/)
- A component's/system's version number has the form X.Y.Z
- X is the *major version number* :
  - Incremented when a release contains a significant, a large or potentially *backward-incompatible* change
  - 0 for initial pre-release development
- Y is the *minor version number* :
  - Incremented when new, minor features are introduced in a *backward-compatible* manner
  - Reset to 0 at each change of major version
- Z is the *patch number* :
  - Incremented when a new build of the software is released due to small bug fixes that keep *backward-compatibility*
  - Reset to 0 at each change of minor version

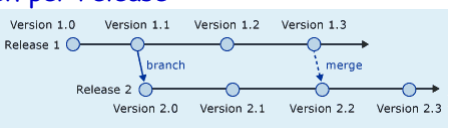
- Example from Wikipedia:

"VersionNumbers" by AzaToth - w:en:Image:VersionNumbers.svg. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:VersionNumbers.svg#mediaviewer/File:VersionNumbers.svg

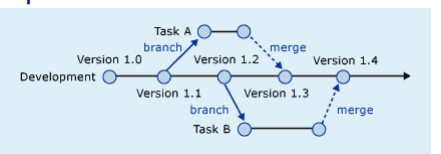


- This is OK for strictly *sequential* developments, but:
- A new version might be derived from *any* previous version
- In practice a more elaborate scheme may be needed for *parallel branches,*
  - See *http://blog.codinghorror.com/software-branching-and-parallel-universes/* and *https://en.wikipedia.org/wiki/Branching_(version_control)*
  - Wikipedia: "Branching is the duplication of an object under revision control so that modifications can happen in parallel along both branches"
  - *A parent branch* might have *child branches*, and development might continue along the *main trunk* as well
  - Branches are often *merged* back into parent or main trunk
  - *codinghorror* shows various scenarios (next slide)

- Some scenarios from the *codinghorror* blog:
  - "Branch per release"



  - "Branch per task"



- *To cope with all this, comprehensive and accurate records must be kept in the configuration/version control system of the relationships between versions*

- Another term: "forking" (especially in GitHub)
  - Paraphrasing GitHub Help:
    "A *fork* is a copy of a project. Forking allows you to freely experiment with changes, or develop a variant system, without affecting the original project"
  - From Wikipedia:
    "A branch not intended to be merged is usually called a *fork*"

Version management

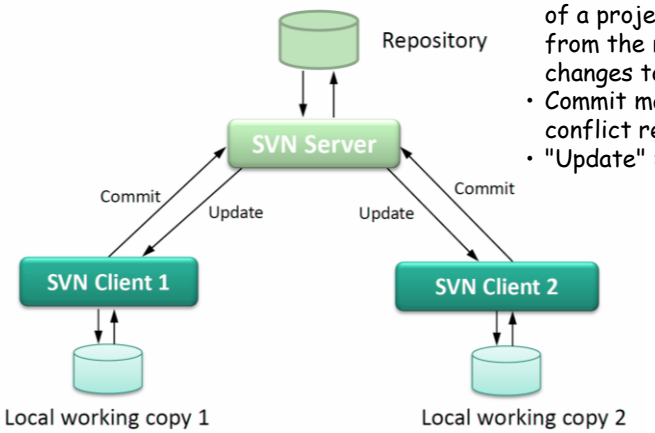
- VCS tools control a *repository* where the different versions of components are held
  - The versions in the repository are *immutable* – that is once entered *they are never changed*
  - The repository allocates new version numbers when an updated component is deposited
  - … linking back to the original version,
  - … and recording information about the update details
- The VCS can report complete change histories for components, and report the differences between versions
- For effective storage management, component "deltas" are computed and recorded at check-in
  - A delta is a *change* in the component
  - Previous versions can be reconstructed, and might not be held explicitly!

VCS and Collaborative working

- VCS tools usually apply *access control mechanisms* and policies:
  - A programmer must *"check a component version out"* of the repository to work on it
  - … and then must *"check it back in"* when the update is complete (*"commit"* the update)
  - The repository allocates new version numbers at check-in
  - … and usually requires information about the *change* represented by the new version
  - The original version is not discarded
  - The VCS knows and records programmer identities with checked in versions

- A component might be *locked* whilst checked-out
  - … to prevent an accidental second checkout followed by parallel updates
- *Or multiple* check-outs might be allowed
  - with *merging* and *conflict resolution* at check-in
- Also it may be possible to request a *copy* of a version without checking it out
  - … perhaps for testing other components
  - *Copies* may *not* be checked in again
- Typically administered as a distributed system
  - With the repository on a central server
  - And individual programmers at their own workstations

Subversion – general schematic of Client-Server system



- The Server organizes a central repository
- Clients keep a local working copy of a project, can fetch updates from the repository and commit changes to it
- Commit may merge and trigger conflict resolution
- "Update" = "checkout"

"SVN Server Client Structure" by L.palm - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:SVN_Server_Client_Structure.png#mediaviewer/File:SVN_Server_Client_Structure.png

Some VCS implementations

- Also known as *source code control systems, source code management systems, software configuration management systems, revision control systems...*
  - See http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- CVS ("Concurrent Version System") is a popular server-based version management tool (originating C1986)
  - http://www.nongnu.org/cvs/
  - It is a powerful VCS, with a low level command line interface needing to be integrated into an IDE

- Apache Subversion: "originally designed to be a better CVS" (from the early 2000s) – now widely used
  - http://subversion.apache.org/
  - "Enterprise-class centralized version control for the masses"
  - Maintains multiple repositories and user accounts
  - Eclipse can interwork with Subversion (eg Subclipse plugin)
  - *We have a Subversion installation*:
    There will be one repository per group,
    shared by all user accounts within the group
- Git: widely used (http://www.git-scm.com/)
  - "Git is a free & open source, distributed version control system... a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. "
  - Good for individual and collaborative work

- Together Architect can use CVS, but also StarTeam (a Microfocus product, https://www.microfocus.com/products/change-management/starteam) and ClearCase (IBM, www.ibm.com/software/awdtools/clearcase/)
  - Paraphrasing Wikipedia:
    "StarTeam … supports branching and 3-way merging, difference analysis, advanced user access and security management, checkpoints, end user and administrator auditing, and customization features. The server is designed for distributed use and supports strong encryption for remote connections. In addition to file versioning, StarTeam also stores requirements, project tasks, change requests, and discussions. All of these can be interlinked to enhance traceability within a project."

- A Microsoft product: Team Foundation Server – a server-based distributed system
  - http://msdn.microsoft.com/en-US/library/aa730884%28v=vs.80%29.aspx
  - "Team Foundation Server (commonly abbreviated to TFS) is a Microsoft product which provides source code management (either via Team Foundation Version Control or Git), reporting, requirements management, project management (for both agile software development and waterfall teams), automated builds, lab management, testing and release management capabilities. It covers the entire Application Lifecycle Management. TFS can be used as a back end to numerous integrated development environments but is designed to provide the most benefit by serving as the back end to Microsoft Visual Studio or Eclipse."
    (Wikipedia summary)

End of lecture