Software Configuration Management

Steven J Zeil

March 7, 2013


Outline


Outline I


Software Configuration Management

- Over time, a software system can exist in many versions:
  - *revisions* created as developers check in changes
  - *configurations* intended for different hardware, operating system, or application environments
  - *releases* issued to users
    - which, if under continued support, may have separate tracks of revisions recording separate bug fixes
- *Software Configuration Management* (SCM) is concerned with all of these


SCM Activities

- Version control
- Build management
- Environment management
- Change management

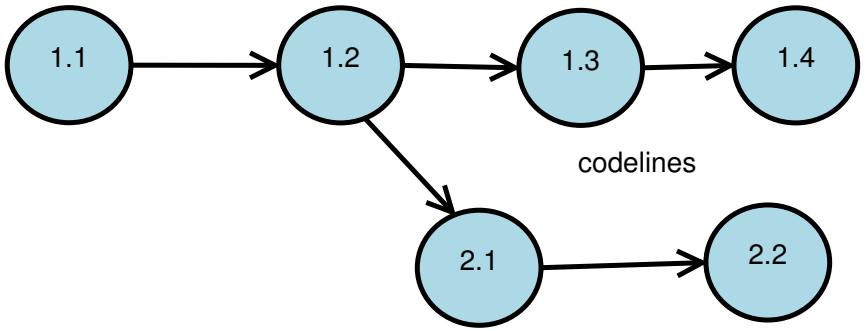
We have seen some tools oriented towards some of these.

- But the broader SCM context may alter how we use some of them

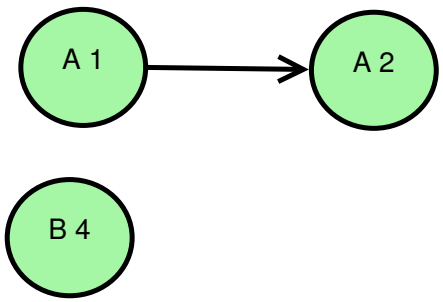

Codelines and Baselines

- A *codeline* is a sequence of revisions of a *configuration item*
  - In essence, a branch
- A *baseline* is a collection of component versions that make up a system.

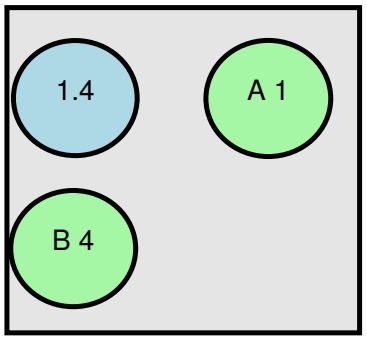

Codelines and Baselines: Example

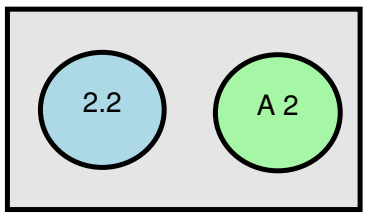


External Libraries and components



Baseline: Windows Release 2





Baseline: Linux Release 3

Baselines

- A major challenge of SCM is coping with multiple baselines that must
  - co-exist and
  - be actively maintained.
- Major issues are
  - deciding when to "freeze" on a version of an imported library
  - tracking the transitive closure of dependencies from libraries that we directly depend upon
  - finding a mutually compatible set of versions among all those external libraries


Environment Management

Coping with the different environments in which the software may need to be installed and/or built.

- Strategies include
  - separate files
    - Easier to manage in the C/C++ world than in Java
  - deltas (patches)
  - conditional compilation
    - Favored in the C/C++ world
    - Harder to support in Java world
  - dynamic loading
    - Common in the Java world
    - Often controlled by "property files" that name modules to be employed for designated tasks in a given installation.


Example: the ODU Extract Project

Metadata extraction system, needed to support

- One release version (thank the heavens)
- Windows, Linux, & Mac platforms
- Choice of 2 OCR programs (or none at all)
  - With local or network access to licensed copies
  - With or without caching of OCR results
- Statistical models trained on different document collections
- Varying client requirements for data post-processing

Problem was not so much the number of choices as the combinatorics.

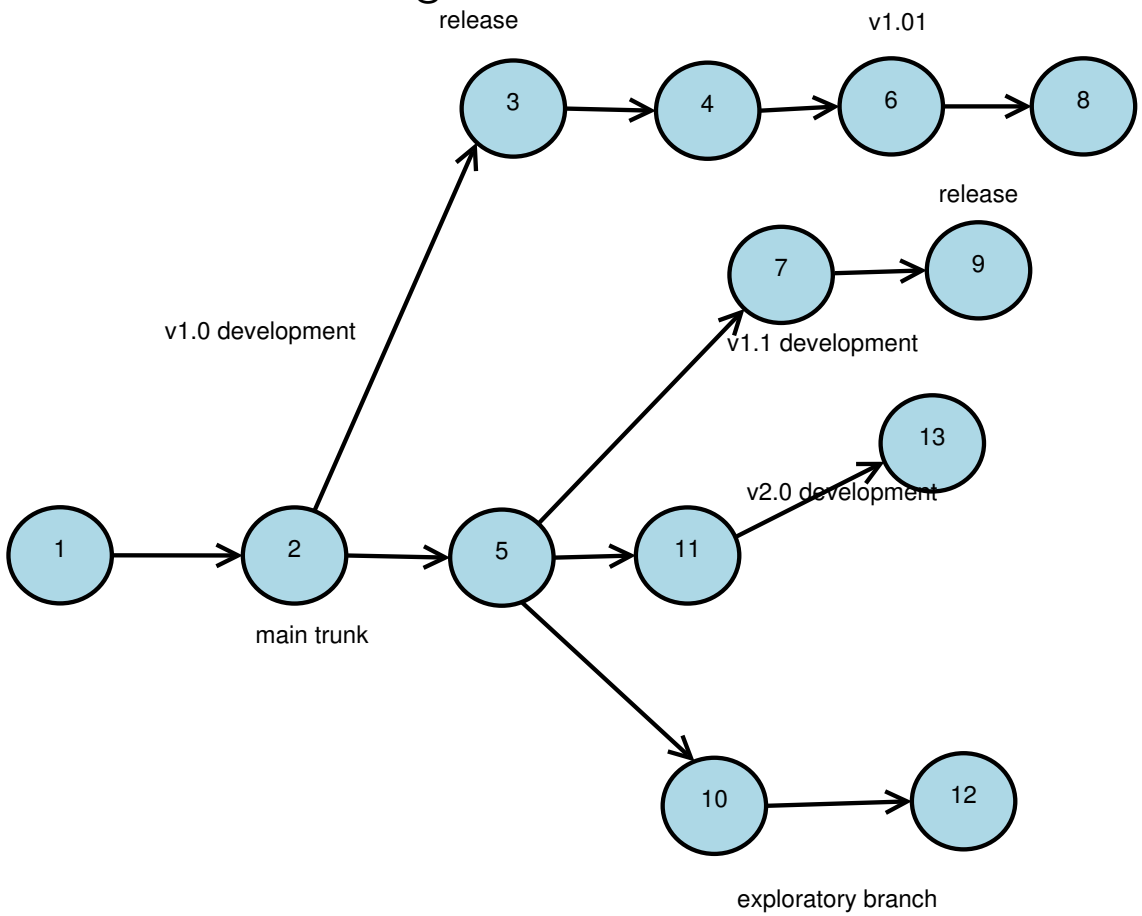

Outline I


Baselines Managed by Build Manager

- Build manager is told what external libraries are needed
  - including desired versions
- Build manager may be responsible for collecting desired versions of both external and internal code from version control.
- Build files are managed as part of each version.


Codelines == Branches

- Main trunk moves forward in time
- Each planned release is a branch from the trunk
  - continues forward through its maintenance lifetime



Change Management

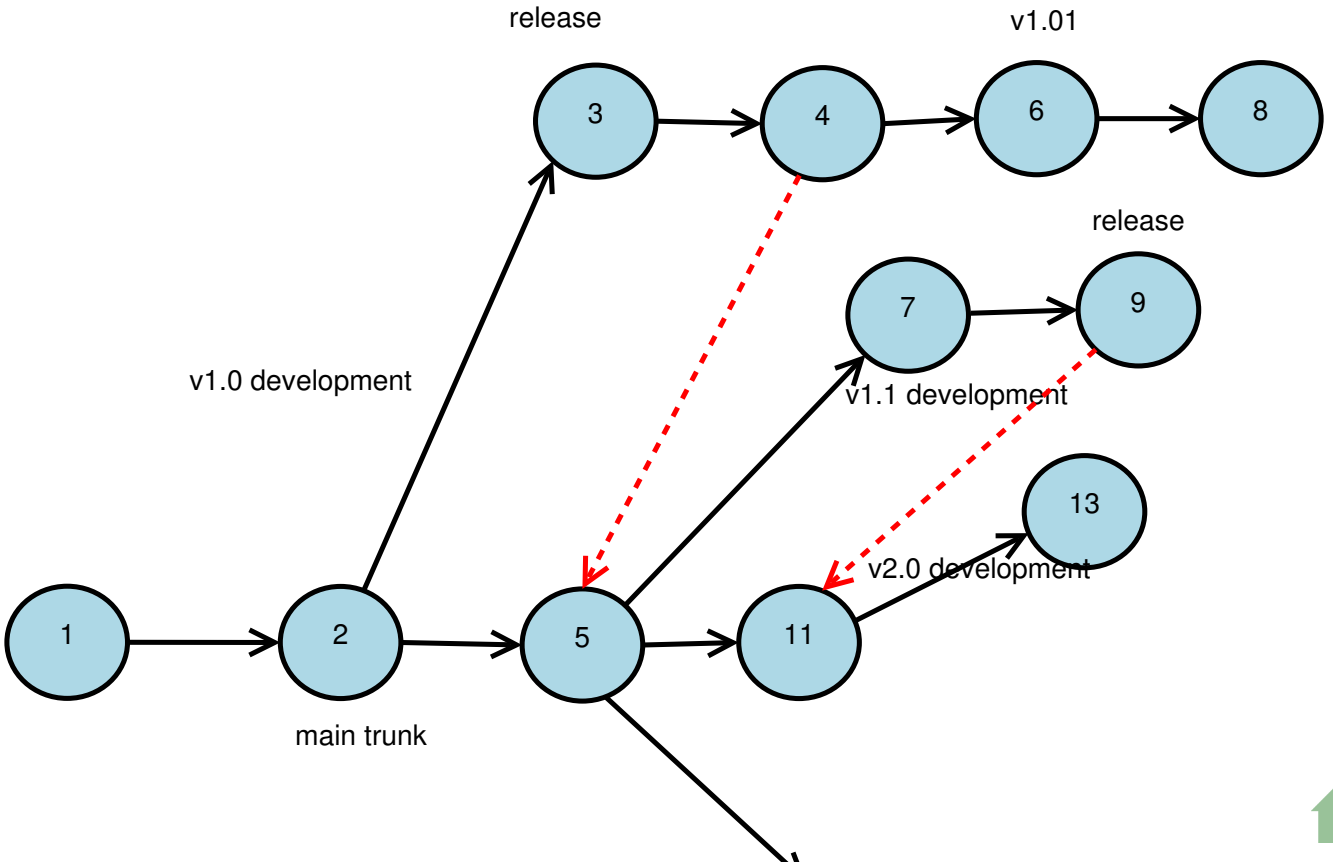
In large organizations, changes are approved by a Change Management Board.
E.g., the team working in an exploratory branch has demonstrated an attractive new feature.

- Should we adopt it?
  - If so, which of the version code lines should it be added to?


Change Propagation I

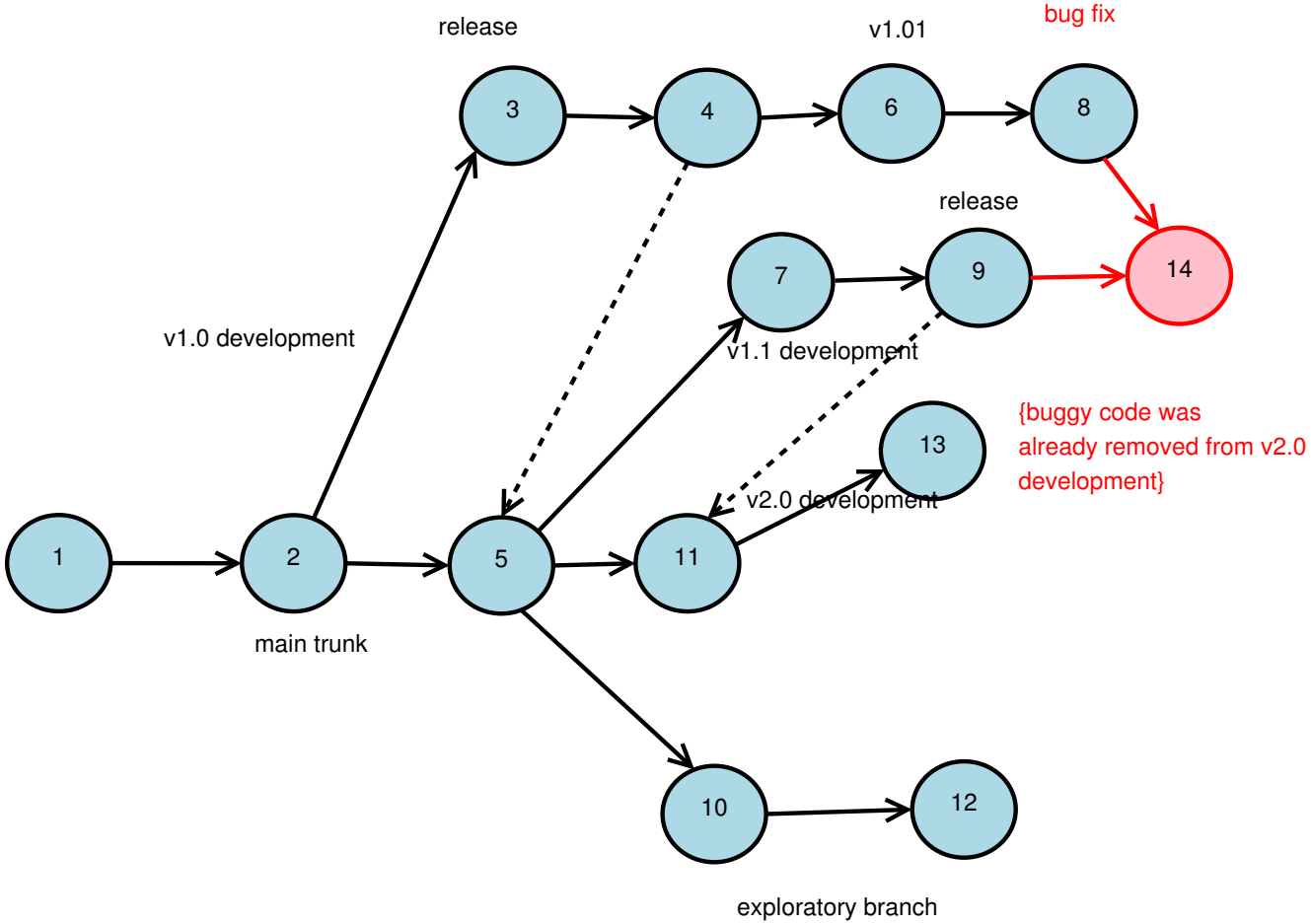
Even in smaller projects, the issue of *change propagation* across code lines needs to be kept in mind.

- The whole "main trunk moves forward" idea presumes that most release changes are synced into the trunk:



Change Propagation II

- As a practical matter, someone has to decide whether bug fixes in older versions can and should be merged into later versions.



Change Propagation III


Simpler Project Structure

In current practice,

- Large projects composed of multiple subprojects are discouraged
- in favor of smaller, independent projects (e.g., one per original subproject)
  - A common rule of thumb is that one project should produce one product (e.g., a single Jar file)


Simpler Project Structure

In current practice,

- Large projects composed of multiple subprojects are discouraged
- in favor of smaller, independent projects (e.g., one per original subproject)
  - A common rule of thumb is that one project should produce one product (e.g., a single Jar file)
  - Plus, perhaps, a source distribution.


Simpler Project Structure

In current practice,

- Large projects composed of multiple subprojects are discouraged
- in favor of smaller, independent projects (e.g., one per original subproject)
  - A common rule of thumb is that one project should produce one product (e.g., a single Jar file)
  - Plus, perhaps, a source distribution.
    - and those are increasingly being replaced by centralized VC repositories