Configuration Management

Lecture 29

Topics covered

- Version management
- System building
- Change management
- Release management

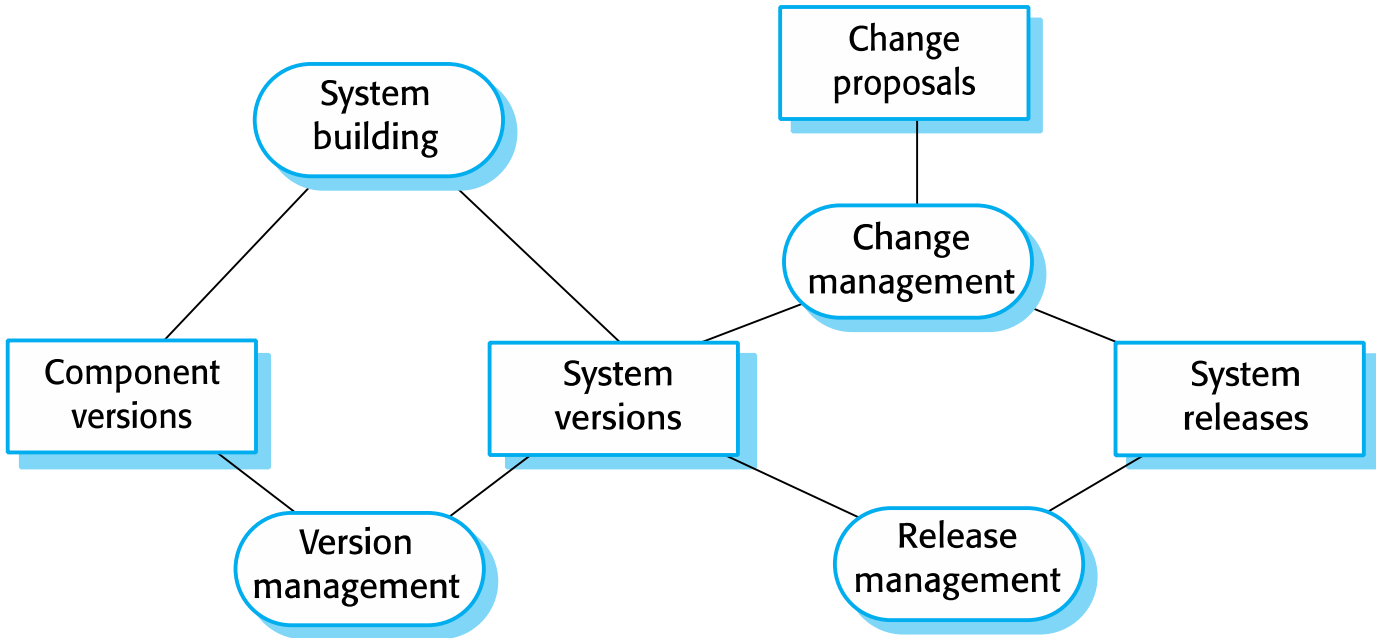
Configuration management

- Software systems are constantly changing during development and use.
- Configuration management (CM) is concerned with the policies, processes and tools for managing changing software systems.
- You need CM because it is easy to lose track of what changes and component versions have been incorporated into each system version.
- CM is essential for team projects to control changes made by different developers

CM activities

- Version management
  - Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.
- System building
  - The process of assembling program components, data and libraries, then compiling these to create an executable system.
- Change management
  - Keeping track of requests for changes to the software from customers and developers, working out the costs and impact of changes, and deciding the changes should be implemented.
- Release management
  - Preparing software for external release and keeping track of the system versions that have been released for customer use.

Configuration management activities



Agile development and CM

- Agile development, where components and systems are changed several times per day, is impossible without using CM tools.
- The definitive versions of components are held in a shared project repository and developers copy these into their own workspace.
- They make changes to the code then use system building tools to create a new system on their own computer for testing. Once they are happy with the changes made, they return the modified components to the project repository.

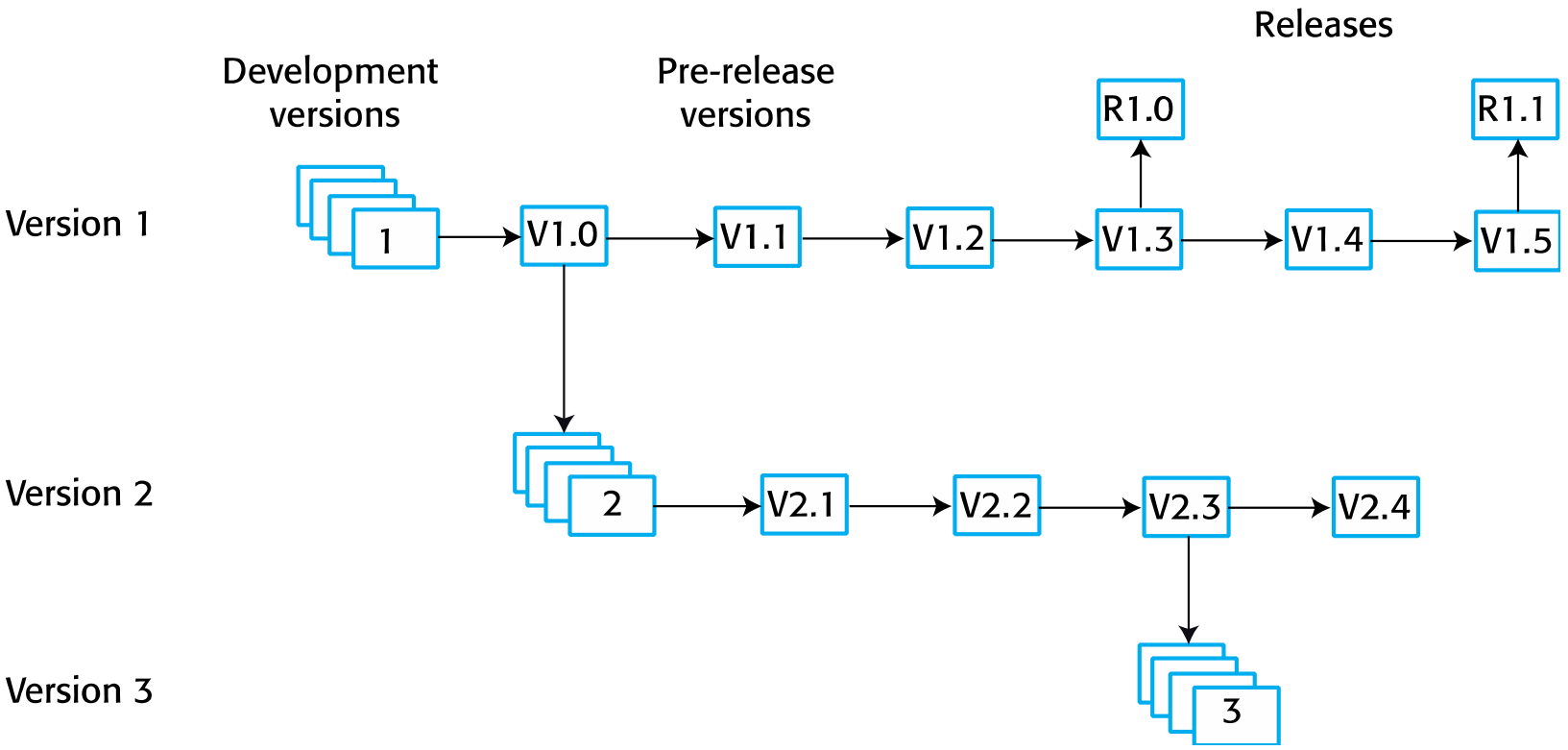
Development phases

- A development phase where the development team is responsible for managing the software configuration and new functionality is being added to the software.
- A system testing phase where a version of the system is released internally for testing.
  - No new system functionality is added. Changes made are bug fixes, performance improvements and security vulnerability repairs.
- A release phase where the software is released to customers for use.
  - New versions of the released system are developed to repair bugs and vulnerabilities and to include new features.

Multi-version systems

- For large systems, there is never just one ‘working’ version of a system.
- There are always several versions of the system at different stages of development.
- There may be several teams involved in the development of different system versions.

Multi-version system development



CM terminology

Term	Explanation
Baseline	A baseline is a collection of component versions that make up a system. Baselines are controlled, which means that the versions of the components making up the system cannot be changed. This means that it is always possible to recreate a baseline from its constituent components.
Branching	The creation of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently.
Codeline	A codeline is a set of versions of a software component and other configuration items on which that component depends.
Configuration (version) control	The process of ensuring that versions of systems and components are recorded and maintained so that changes are managed and all versions of components are identified and stored for the lifetime of the system.
Configuration item or software configuration item (SCI)	Anything associated with a software project (design, code, test data, document, etc.) that has been placed under configuration control. There are often different versions of a configuration item. Configuration items have a unique name.
Mainline	A sequence of baselines representing different versions of a system.

CM terminology

Term	Explanation
Merging	The creation of a new version of a software component by merging separate versions in different codelines. These codelines may have been created by a previous branch of one of the codelines involved.
Release	A version of a system that has been released to customers (or other users in an organization) for use.
Repository	A shared database of versions of software components and meta-information about changes to these components.
System building	The creation of an executable system version by compiling and linking the appropriate versions of the components and libraries making up the system.
Version	An instance of a configuration item that differs, in some way, from other instances of that item. Versions always have a unique identifier.
Workspace	A private work area where software can be modified without affecting other developers who may be using or modifying that software.

Version management

Version management

- Version management (VM) is the process of keeping track of different versions of software components or configuration items and the systems in which these components are used.
- It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
- Therefore version management can be thought of as the process of managing codelines and baselines.

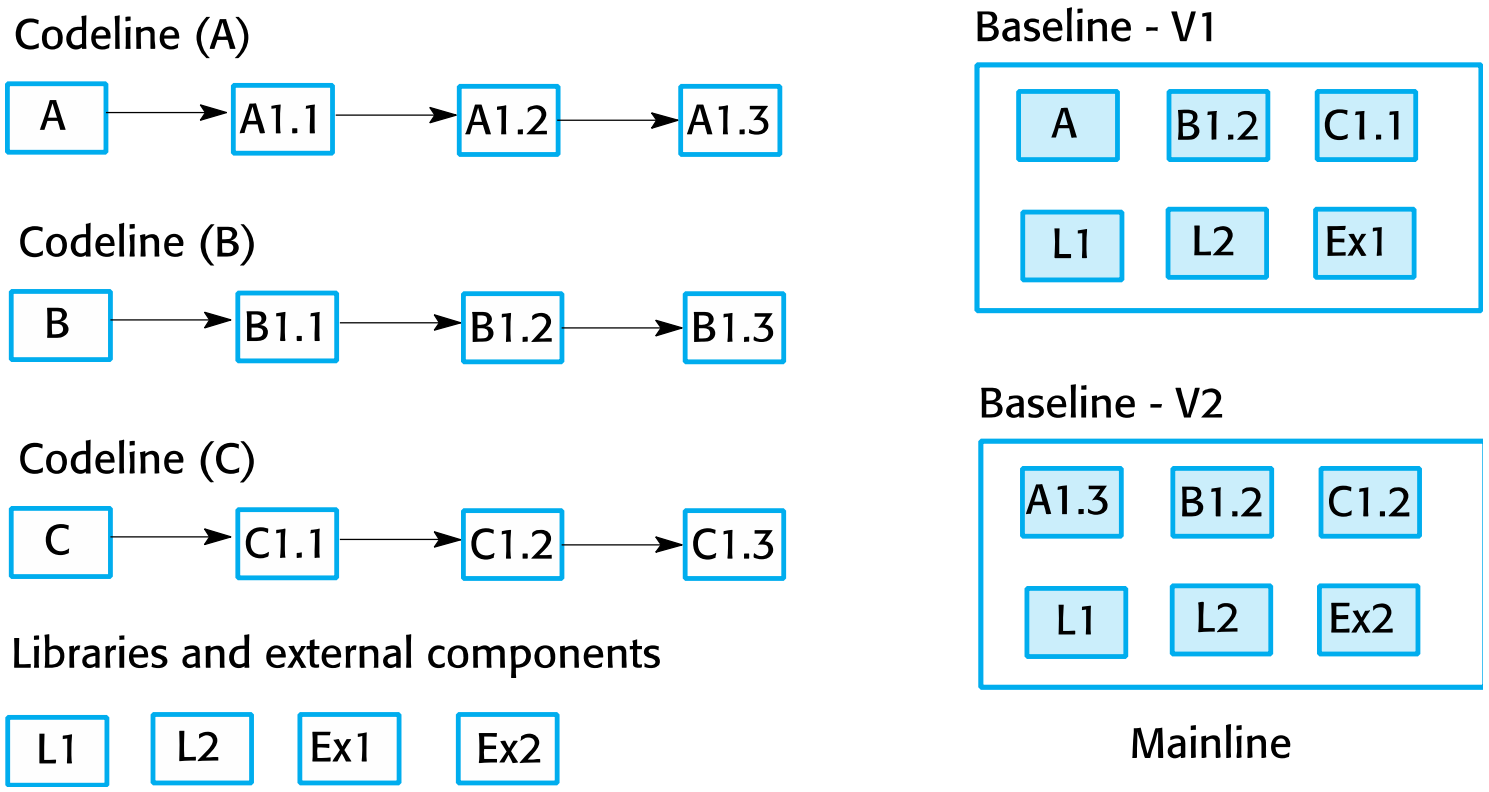
Codelines and baselines

- A codeline is a sequence of versions of source code with later versions in the sequence derived from earlier versions.
- Codelines normally apply to components of systems so that there are different versions of each component.
- A baseline is a definition of a specific system.
- The baseline therefore specifies the component versions that are included in the system plus a specification of the libraries used, configuration files, etc.

Baselines

- Baselines may be specified using a configuration language, which allows you to define what components are included in a version of a particular system.
- Baselines are important because you often have to recreate a specific version of a complete system.
  - For example, a product line may be instantiated so that there are individual system versions for different customers. You may have to recreate the version delivered to a specific customer if, for example, that customer reports bugs in their system that have to be repaired.

Codelines and baselines



Version control systems

- Version control (VC) systems identify, store and control access to the different versions of components. There are two types of modern version control system
  - Centralized systems, where there is a single master repository that maintains all versions of the software components that are being developed. Subversion is a widely used example of a centralized VC system.
  - Distributed systems, where multiple versions of the component repository exist at the same time. Git is a widely-used example of a distributed VC system.

Key features of version control systems

- Version and release identification
- Change history recording
- Support for independent development
- Project support
- Storage management

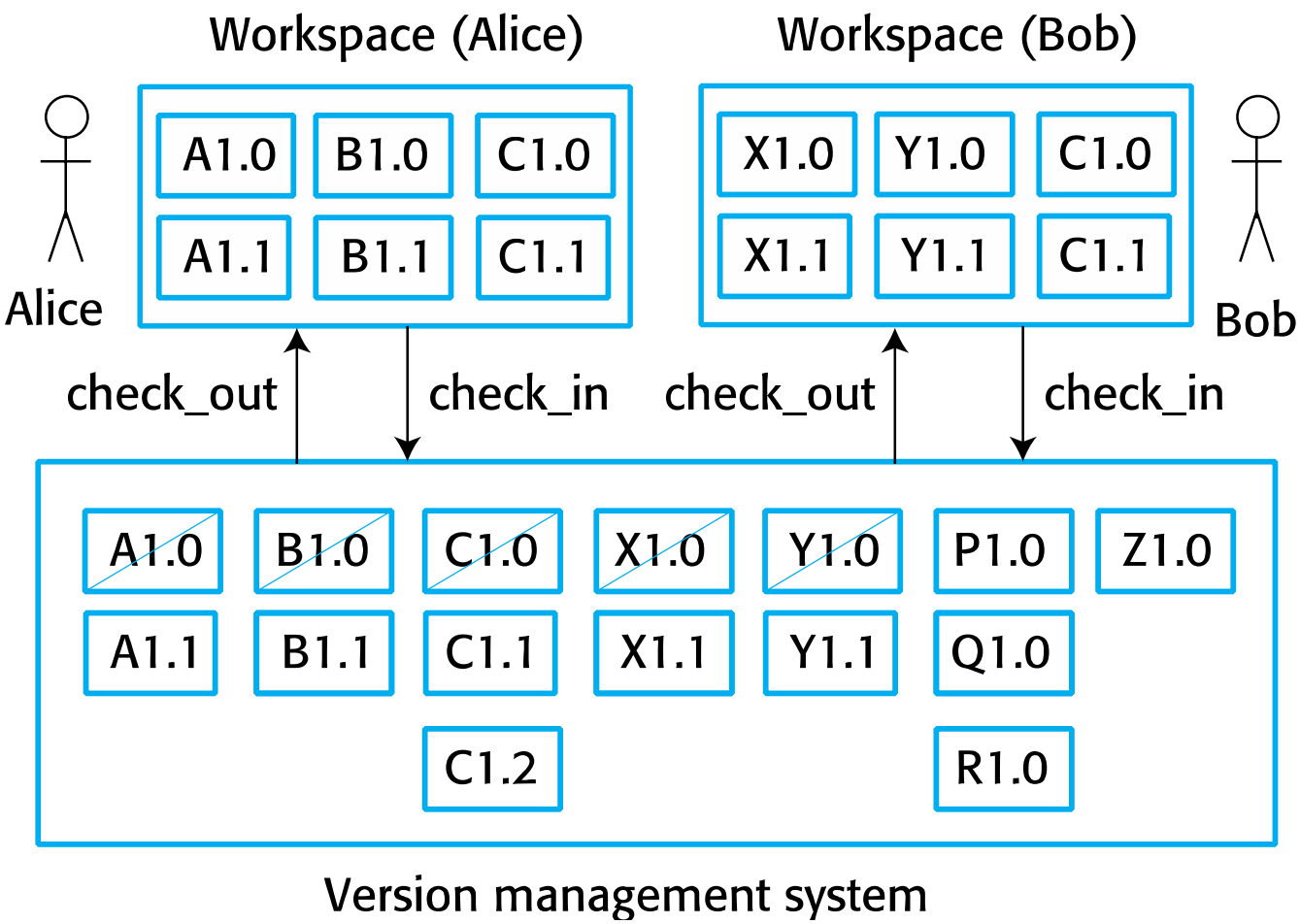
Public repository and private workspaces

- To support independent development without interference, version control systems use the concept of a project repository and a private workspace.
- The project repository maintains the ‘master’ version of all components. It is used to create baselines for system building.
- When modifying components, developers copy (check-out) these from the repository into their workspace and work on these copies.
- When they have finished their changes, the changed components are returned (checked-in) to the repository.

Centralized version control

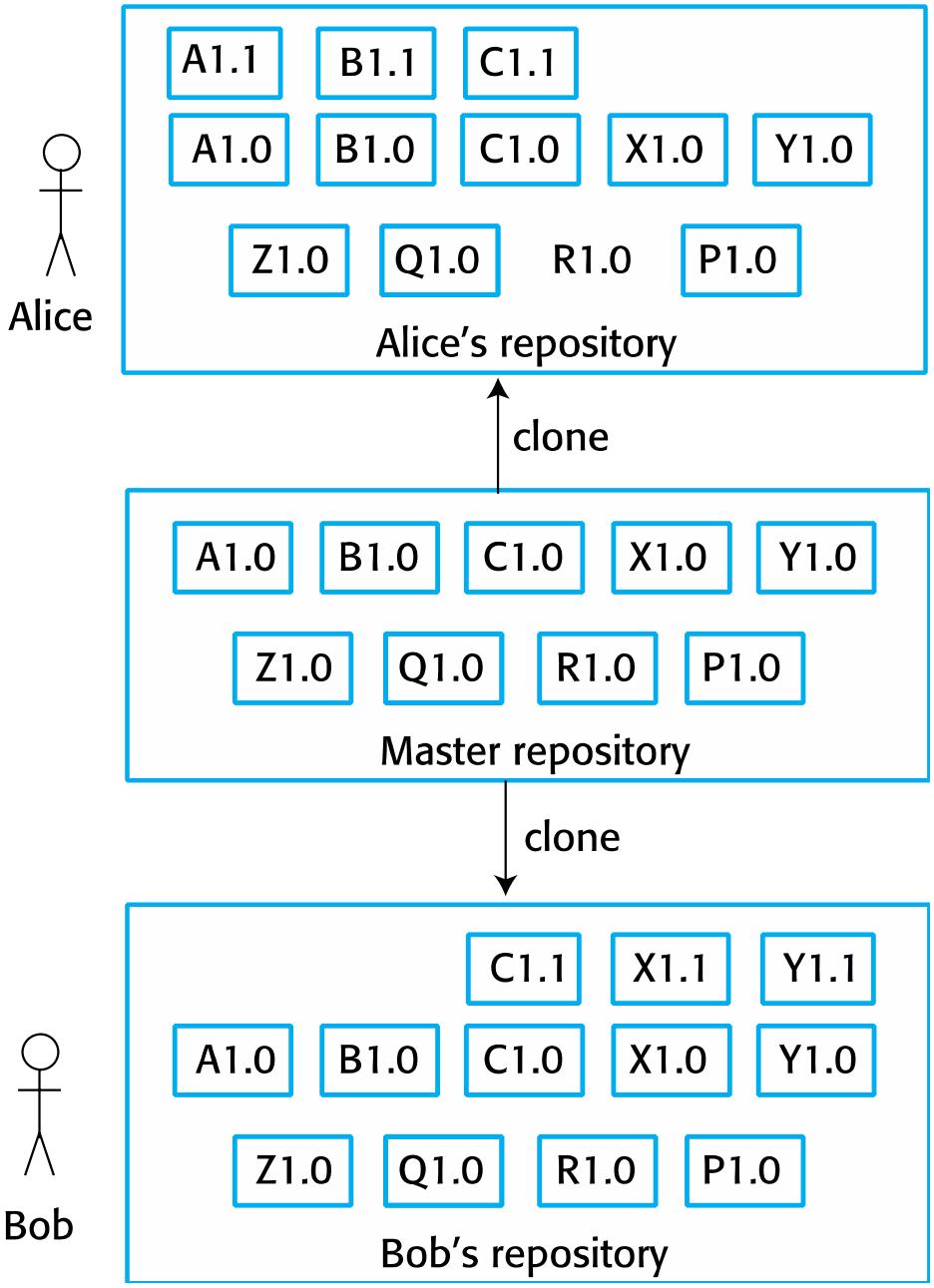
- Developers check out components or directories of components from the project repository into their private workspace and work on these copies in their private workspace.
- When their changes are complete, they check-in the components back to the repository.
- If several people are working on a component at the same time, each check it out from the repository. If a component has been checked out, the VC system warns other users wanting to check out that component that it has been checked out by someone else.

Repository Check-in/Check-out



Distributed version control

- A ‘master’ repository is created on a server that maintains the code produced by the development team.
- Instead of checking out the files that they need, a developer creates a clone of the project repository that is downloaded and installed on their computer.
- Developers work on the files required and maintain the new versions on their private repository on their own computer.
- When changes are done, they ‘commit’ these changes and update their private server repository. They may then ‘push’ these changes to the project repository.



Repository cloning

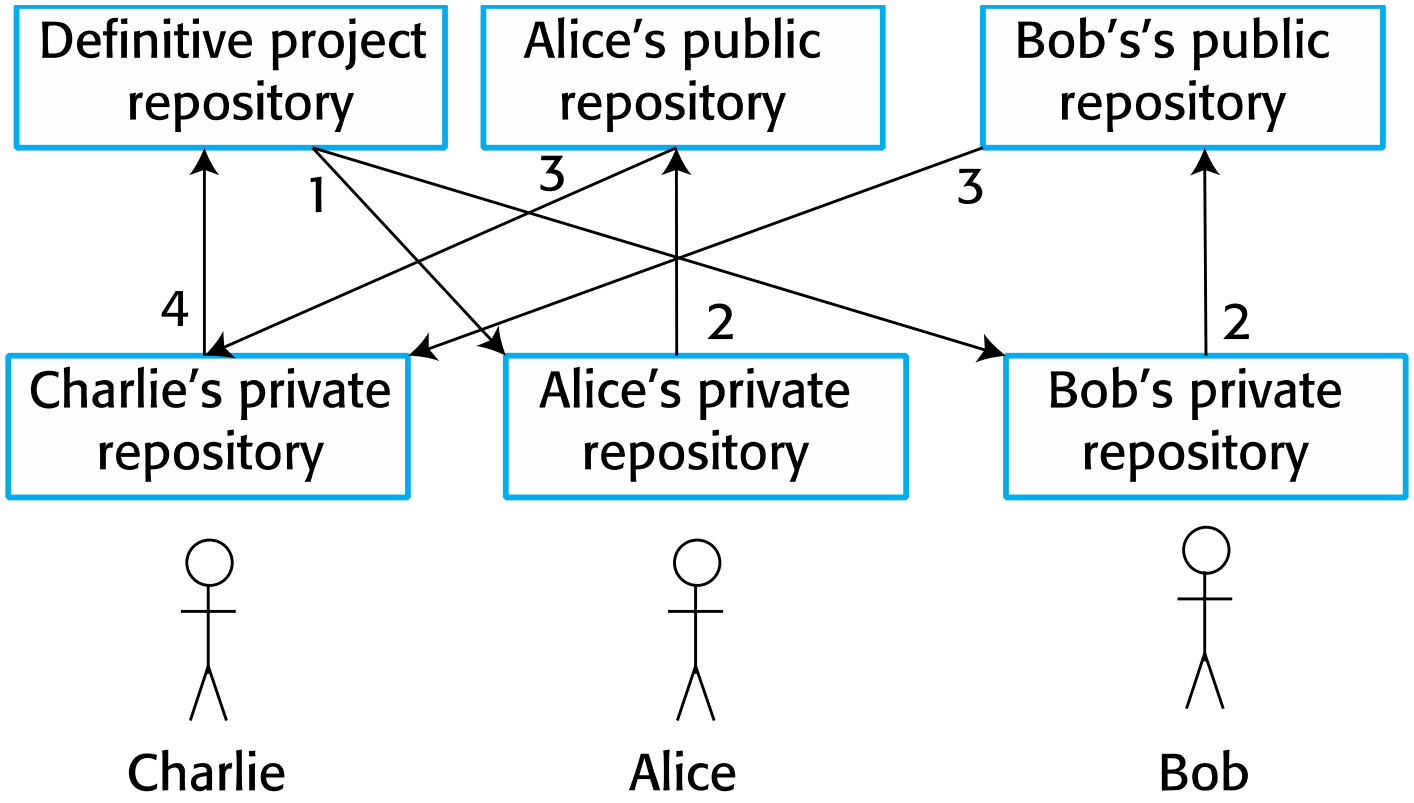
Benefits of distributed version control

- It provides a backup mechanism for the repository.
  - If the repository is corrupted, work can continue and the project repository can be restored from local copies.
- It allows for off-line working so that developers can commit changes if they do not have a network connection.
- Project support is the default way of working.
  - Developers can compile and test the entire system on their local machines and test the changes that they have made.

Open source development

- Distributed version control is essential for open source development.
  - Several people may be working simultaneously on the same system without any central coordination.
- As well as a private repository on their own computer, developers also maintain a public server repository to which they push new versions of components that they have changed.
  - It is then up to the open-source system ‘manager’ to decide when to pull these changes into the definitive system.

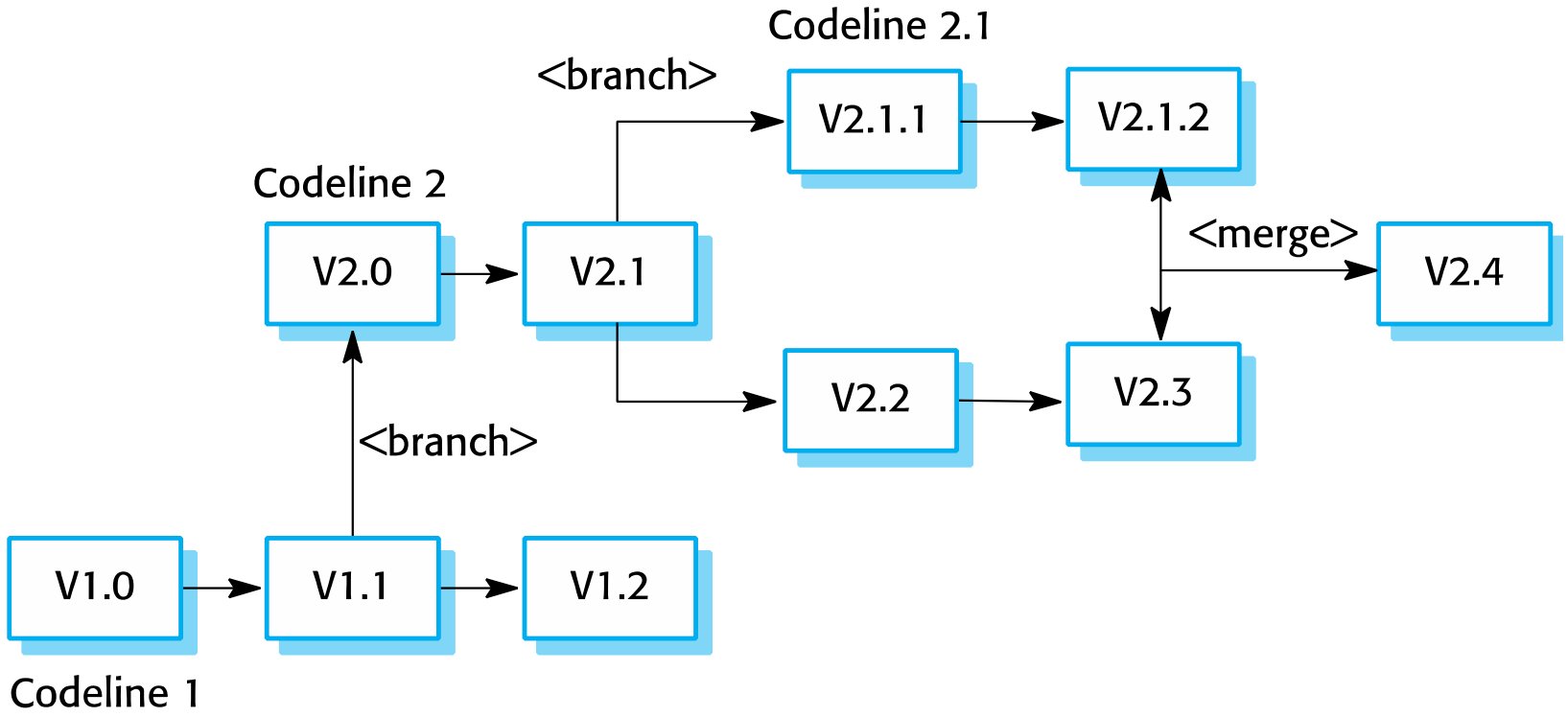
Open-source development



Branching and merging

- Rather than a linear sequence of versions that reflect changes to the component over time, there may be several independent sequences.
  - This is normal in system development, where different developers work independently on different versions of the source code and so change it in different ways.
- At some stage, it may be necessary to merge codeline branches to create a new version of a component that includes all changes that have been made.
  - If the changes made involve different parts of the code, the component versions may be merged automatically by combining the deltas that apply to the code.

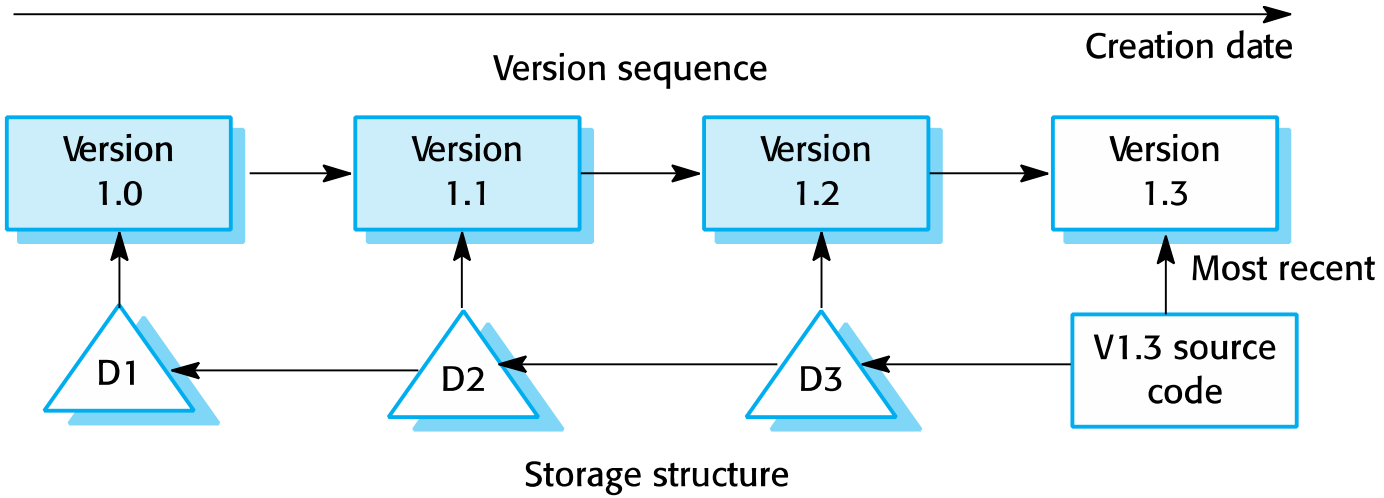
Branching and merging



Storage management

- When version control systems were first developed, storage management was one of their most important functions.
- Disk space was expensive and it was important to minimize the disk space used by the different copies of components.
- Instead of keeping a complete copy of each version, the system stores a list of differences (deltas) between one version and another.
  - By applying these to a master version (usually the most recent version), a target version can be recreated.

Storage management using deltas



Storage management in Git

- As disk storage is now relatively cheap, Git uses an alternative, faster approach.
- Git does not use deltas but applies a standard compression algorithm to stored files and their associated meta-information.
- It does not store duplicate copies of files. Retrieving a file simply involves decompressing it, with no need to apply a chain of operations.
- Git also uses the notion of packfiles where several smaller files are combined into an indexed single file.