CSIT214/CSIT883 IT Project Management

Change and version control 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.

✧ Why do we need CM?

   ✧ 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

✧ **Change management**

▪ Keeping track of <u>requests</u> for changes to the software from customers and developers, working out the costs and <u>impact</u> of changes, and <u>deciding</u> the changes that should be implemented.

✧ **Version management**

▪ Keeping track of the <u>multiple versions</u> of system components and ensuring that changes made to components by different developers **do not interfere** with each other.

✧ System building

▪ The process of <u>assembling</u> program components, data and libraries, then <u>compiling</u> these to create an executable system.

✧ Release management

▪ Preparing software for <u>external release</u> and keeping track of the system versions that have been released for customer use.

Change management

- Organizational needs and **requirements change** during the lifetime of a system, **bugs** have to be repaired and systems have to **adapt to changes in their environment**.

- Change management is intended to ensure that system evolution is a **managed process** and that priority is given to the most urgent and cost-effective changes.

- The change management process is concerned with:
    - analyzing the costs and benefits of proposed changes,
    - approving those changes that are worthwhile and
    - tracking which components in the system have been changed.
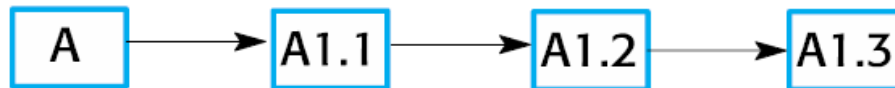
Version management

✧ 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 (e.g. a class or a file) 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.
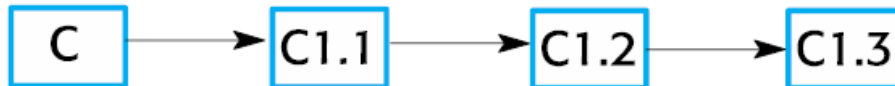
Codelines and baselines

Codeline (A)

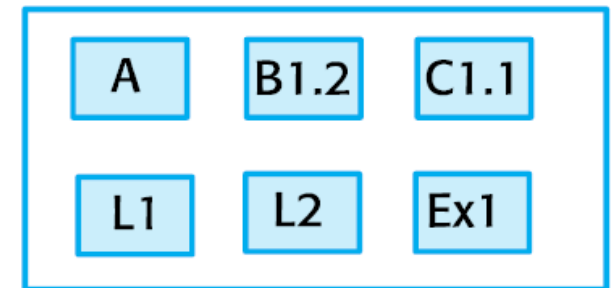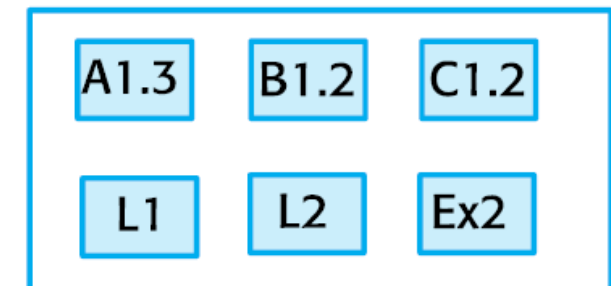A → A1.1 → A1.2 → A1.3

Codeline (B)

B → B1.1 → B1.2 → B1.3

Codeline (C)

C → C1.1 → C1.2 → C1.3

Libraries and external components

L1   L2   Ex1   Ex2

Baseline - V1

A   B1.2   C1.1
L1   L2   Ex1

Baseline - V2

A1.3   B1.2   C1.2
L1   L2   Ex2

Mainline

Version control management 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 (e.g. Subversion and CVS)

  - **Distributed systems**, where multiple versions of the component repository exist at the same time. (e.g. Git)
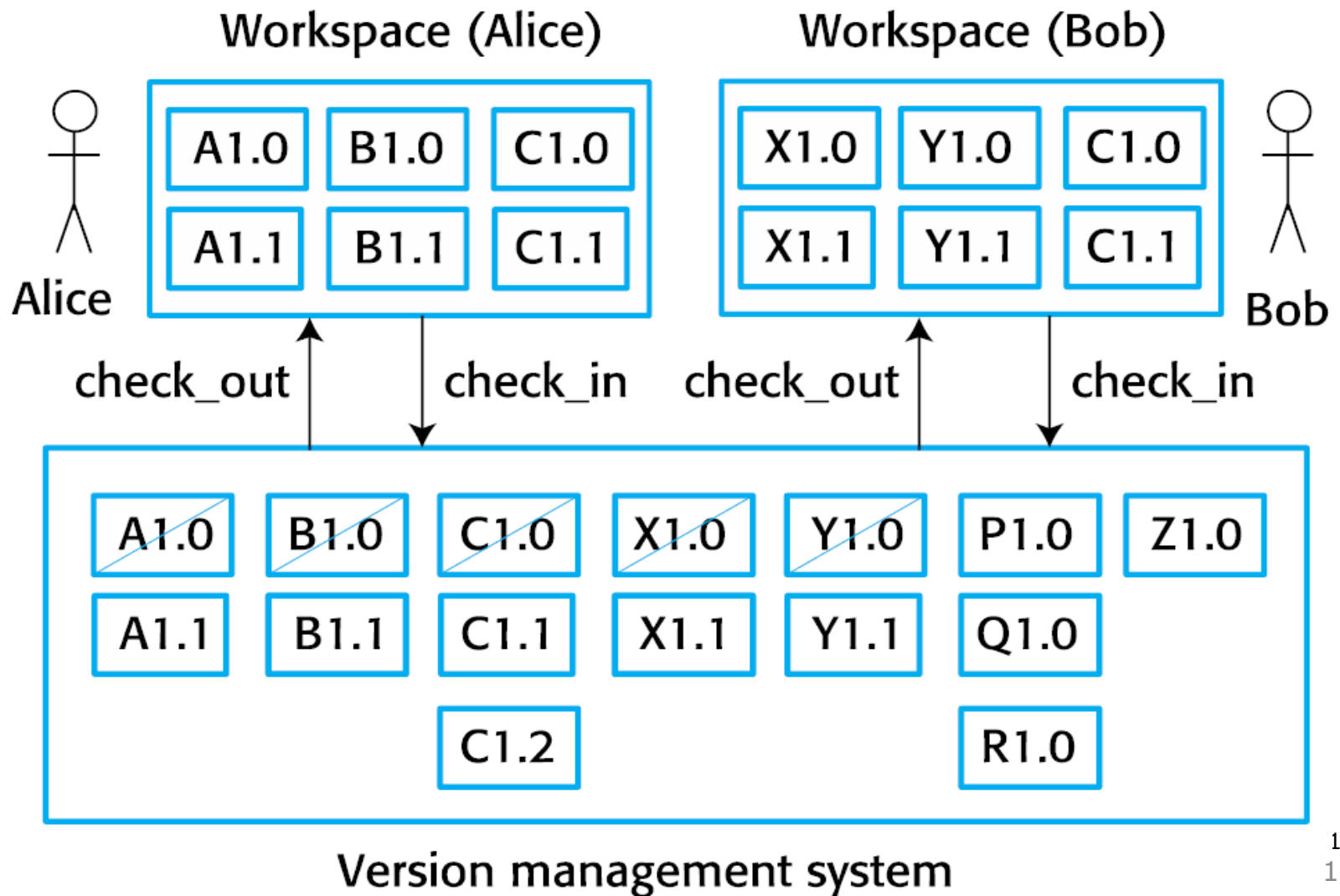
Project repository and private workspaces

- ✧ 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 (checkout) these from the repository into their private 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

Workspace (Alice)

A1.0	B1.0	C1.0
A1.1	B1.1	C1.1

Alice

Workspace (Bob)

X1.0	Y1.0	C1.0
X1.1	Y1.1	C1.1

Bob

check_out   check_in   check_out   check_in

Version management system

A1.0	B1.0	C1.0	X1.0	Y1.0	P1.0	Z1.0
A1.1	B1.1	C1.1	X1.1	Y1.1	Q1.0	
		C1.2			R1.0	

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
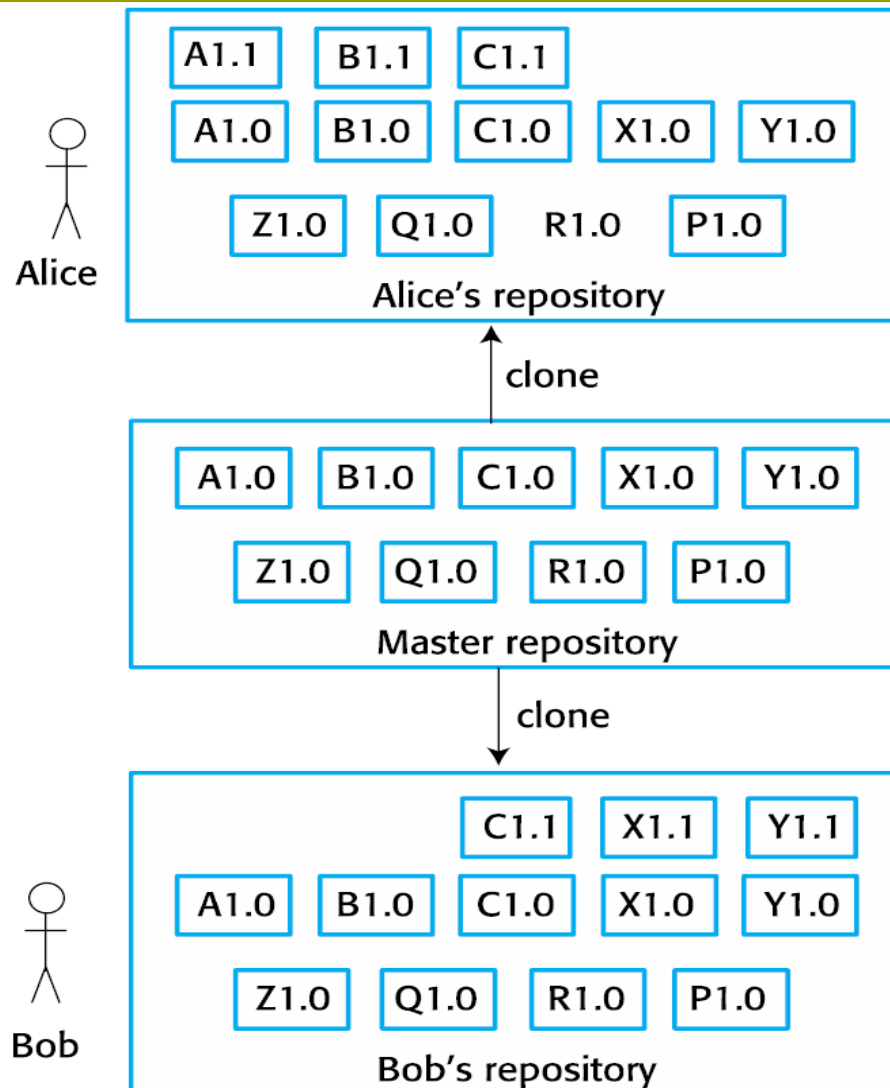
Benefits of distributed version control

✧ 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.

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 (**branches**).

- 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 differences that apply to the code.

Branching and merging