Configuration Management Tools

- **Configuration management tools** support is essential for configuration management.

- **CM tools** are used to
  - store versions of system components,
  - build systems from these components,
  - track the releases of system versions to customers, and
  - keep track of change proposals.

- **CM tools** range from
  - **simple tools** that support a single configuration management task, such as bug tracking,
  - to **integrated environments tools** that support all configuration management activities.

Configuration Management (CM) Terminologies

Term	Explanation
Baseline	A collection of component versions that make up a system. Baselines are controlled, which means that the component versions used in the baseline cannot be changed. It is always possible to re-create 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 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. Configuration items always have a unique identifier.
Mainline	A sequence of baselines representing different versions of a system.

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 should 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.

(1) Version Control Systems

- **Version control (VC) systems**
  - identify, store, and control access to the different versions of components.

(1) Codeline Version Control:

- A codeline is **a sequence of versions** of source code, with later versions in the sequence derived from earlier versions.

(2) Baseline Version Control:

- A baseline is a **definition of a specific system**.
- The baseline **specifies the component versions** that are included in the system and identifies the **libraries used**, **configuration files**, and **other system information**.
- Baselines are important because from which to re-create an individual version of a system.

(3) Mainline Version Control:

- The **mainline** is **a sequence of system versions developed from an original baseline**.

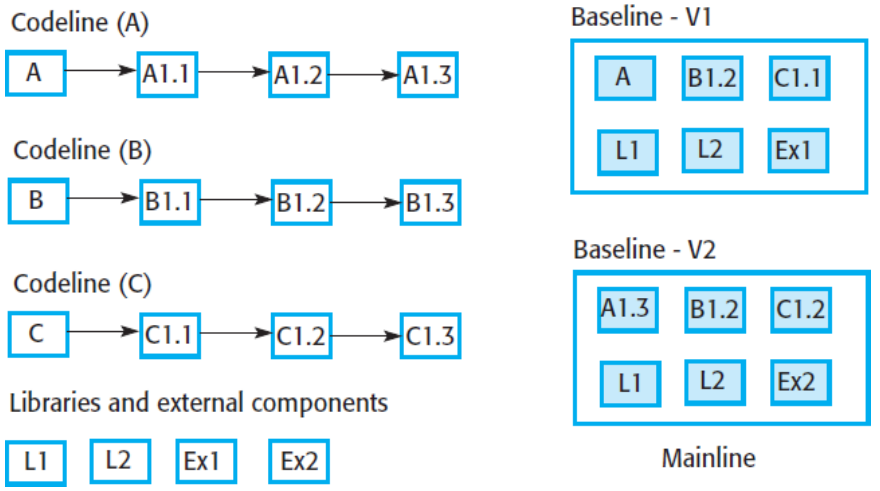

Figure 3.12: the difference between Codelines, Mainline and baselines CM technologies

- different baselines use different versions of the components from each codeline.
- the shaded boxes representing components in the baseline

Types of Version Control Systems

- **Key Features of Version Control Systems**
- 1. Version and Release identification
  - assigned unique identifiers when they are submitted to the system.
- 2. Change history recording
  - keeps records of the changes that have been made by tagging components with keywords describing the changes made.
- 3. Independent development
  - ensures that changes made to a component by different developers do not interfere.
- 4. Project support
  - support the development of several projects, which share components.
- 5. Storage management
  - ensure that duplicate copies of identical files are not maintained.

(1) A Centralized Version Control System

- **Centralized systems**, where a single master repository maintains all versions of the software components that are being developed.
- **Check-out from the Repository**
  - **In centralized systems, Developers check out** components from the project repository **into** their private workspace and work on these copies in their private workspace**.**
- **Check-in to the Repository**
  - When **their changes are complete, they check-in the components** back to the repository.
  - This creates **a new component version that may then be shared**.
- If two or more people are working on a component **at the same time**,
  - each must check out the component from the repository.
- If a component has been checked out**,**
  - ***the version control system warns other users wanting to check out that component that it has been checked out by someone else.***
- When the modified components are checked in,
  - The version control system will also ensure that ***the different versions are assigned different version identifiers and are stored separately***.

(1) A Centralized Version Control System

- **Alice** has **checked out versions A1.0, B1.0, and C1.0**. She has worked on these versions and has **created new versions A1.1, B1.1, and C1.1**. She checks these new versions into the repository.
- **Bob checks out versions X1.0, Y1.0, and C1.0**. He creates new versions of these components and checks them back in to the repository.
- **Alice** has already created a new version of C, while **Bob** has been working on it.
- His check-in therefore creates another version C1.2, so that *Alice's changes are not overwritten*.

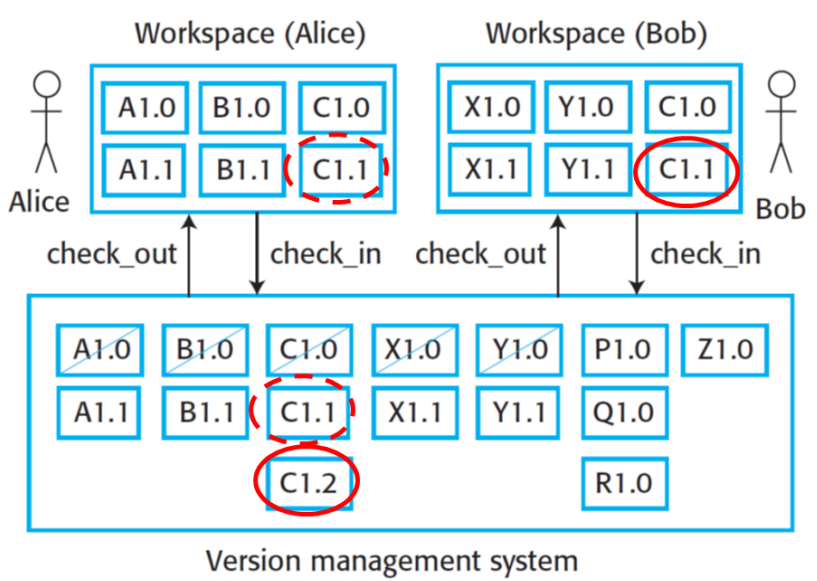

Figure 3.13: Check-in and check-out from a centralized version repository

(2) A Distributed Version Control System

- **Distributed systems**, where multiple versions of the component repository exist at the same time.
- **In a distributed VC system**, such as ***Git***.
  - A ¯master‖ repository is created on a server that maintains the code produced by the development team.
- **“clone” the project repository**
  - A developer **creates a clone of the project repository** that is downloaded and installed on his or her computer.
  - Developers work on the files required and maintain the new versions on **their private repository on their own computer**. When they have finished making changes, they ¯commit‖ these changes and update their **private server repository**.
  -
- **“push” these changes to the project repository**
  - Developers may then **“push” these changes to the project repository or tell the integration manager that changed versions are available.**
- **“pull” these files into the project repository**
  - He or she may then ¯pull‖ these files to the project repository
- In this example, **both Bob and Alice have cloned the project repository** and have updated files. They **have not yet pushed** these **back** to the project repository.

(2) A Distributed Version Control System - Advantages

- 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 offline 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 they have made.
- **Distributed version control** is **essential for open-source development**
  - where several people may be working simultaneously on the same system without any central coordination.
  - There is no way for the open-source system ¯manager‖ to know when changes will be made.
  - In this case, 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**.

2. System Building

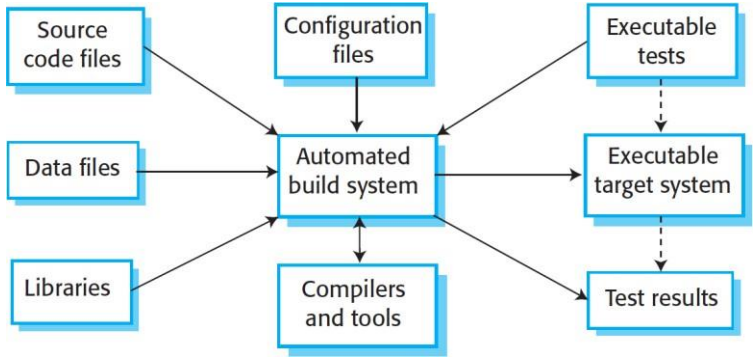

Figure 3.15: System Building

Use an automated build tool to create a system build.

Tools for system integration and building include some or all of the following **features**:

1. Build script generation
2. Version control system integration
3. Minimal recompilation
4. Executable system creation
5. Test automation
6. Reporting
7. Documentation generation

2. System Building - Three different system platform

- System Building is a complex process and three different system platforms may be involved:

1. **The development system**,
   - which includes development tools such as compilers and source code editors.
2. **The build server**,
   - which is used to build definitive, executable versions of the system. This server maintains the definitive versions of a system. All of the system developers check in code to the version control system on the build server for system building.
3. **The target environment**,
   - which is the platform on which the system executes.
   - This may be the same type of computer that is used for the development and build systems.
   - For real-time and embedded systems, the target environment is often smaller and simpler than the development environment (e.g., a cell phone).
   - For large systems, the target environment may include databases and other application systems that cannot be installed on development machines.

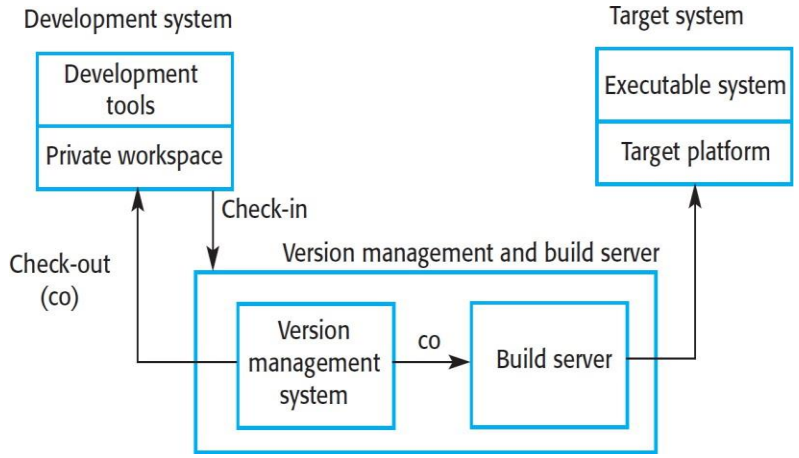

Figure 3.16: Development, build, and target platforms

3. Change Management

- **Change management is**
  - the process of analyzing the costs and benefits of proposed changes, approving those changes that are cost-effective, and tracking which components in the system have been changed.
  -
- Figure 3.17 is **a model of a change management process that shows the main change management activities.**
- This process should come into effect when the software is handed over for release to customers or for deployment within an organization.
- **All change management processes** should include some way of ***checking, costing, and approving changes***

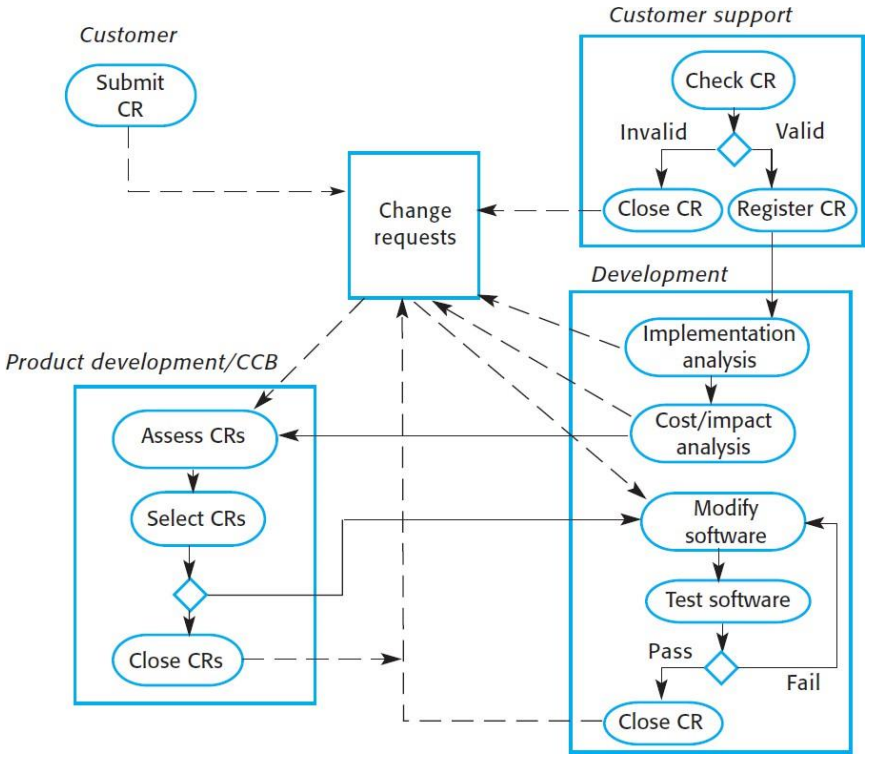

Figure 3.17: Change Management Process